Advanced Operating Systems:
Lab 3 - TCP

Lecturelet 3

Dr Robert Watson

2020-2021

Lab 3 objectives

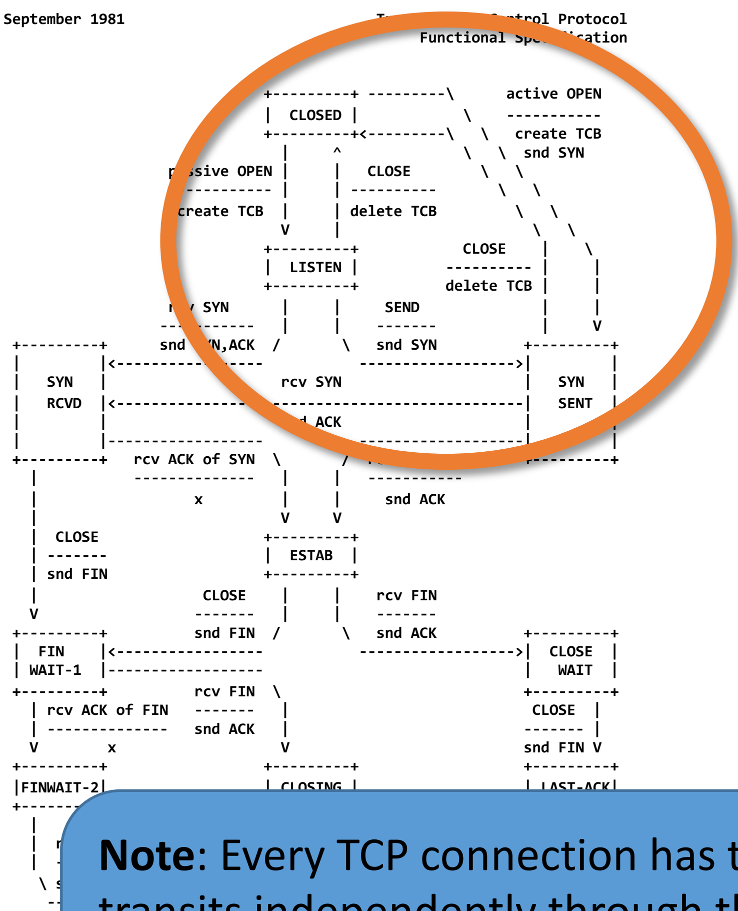
- Further develop tracing, analysis, presentation skills
- Explore the TCP protocol **and** implementation, tracing and analysing internal state and wire-level behaviours
- Experiment with the interactions between TCP and variable network latency; explore:
  - TCP state-machine behaviour and variation (**Part II only**)
  - TCP congestion control behaviour (**L41 only**)
- You are (very) welcome to investigate the other assignment, but you will not receive marks for it
- Gather and analyse data for your third lab submission

New documents

- Advanced Operating Systems: Lab 3 – TCP
- Part II - Advanced Operating Systems: Lab 3 – TCP
- L41 - Advanced Operating Systems: Lab 3 – TCP

- **Important**: The two assignments are substantially different. Please make sure you use the right assignment!

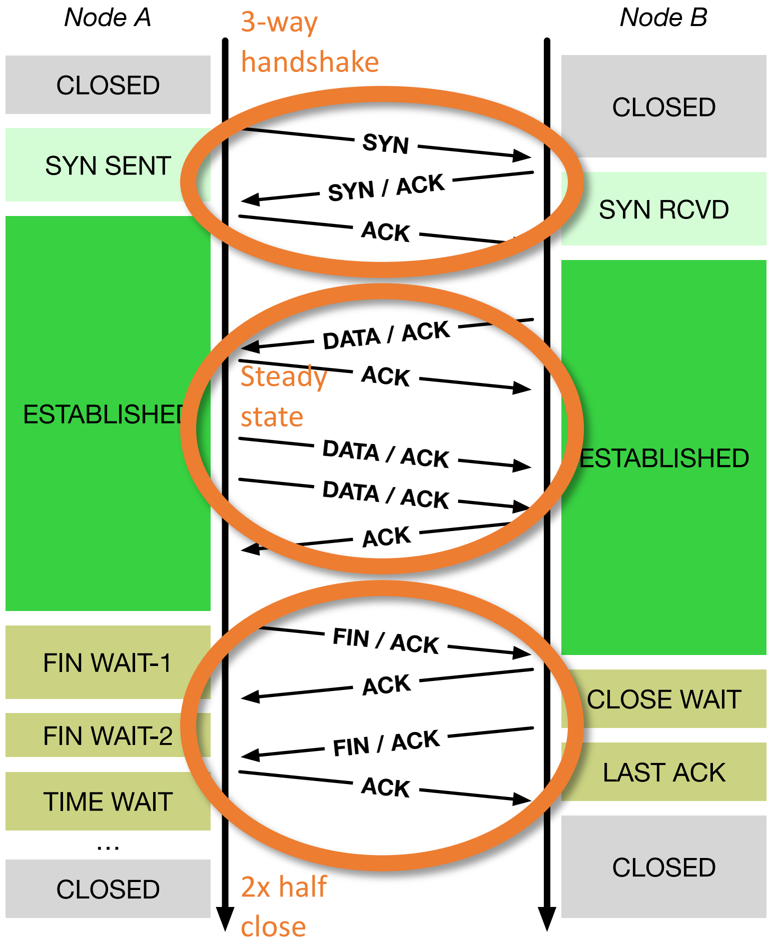
Lecture 6: The Transmission Control Protocol (TCP)



- V. Cerf, K. Dalal, and C. Sunshine, ***Transmission Control Protocol (version 1)***, INWG General Note #72, December 1974.
- In practice: J. Postel, Ed., ***Transmission Control Protocol: Protocol Specification***, RFC 793, September, 1981.

Note: Every TCP connection has two TCBs, one at each endpoint – each of which transits independently through the state machine. When we use loopback connections in our lab assignment, there will be two open sockets, one for each endpoint, and hence two TCP control blocks (tcpcbs). The two endpoints have inverted 4-tuples, so can be identified (with suitable care).

Lecture 6: TCP principles and properties



- Assumptions: Network may delay, (reorder), drop, corrupt IP packets
- TCP implements reliable, ordered, stream transport protocol over IP
- Three-way handshake:
  SYN / SYN-ACK / ACK (mostly!)
- Steady state
  - Sequence numbers ACK'd
  - Round-Trip Time (RTT) measured to time out loss
  - Data retransmitted on loss
  - Flow control via advertised window size in ACKs
  - Congestion control ('fairness') detects congestion via loss (and, recently, via delay: BBR)
- NB: "Half close" allows communications in one direction to end while the other continues

TCP in the IPC benchmark

```
root@rpi4-000:/data/ipc # ./ipc-benchmark
ipc-benchmark [-Bgjqsv] [-b buffersize] [-i pipe|local|tcp] [-n iterations]
    [-p tcp_port] [-P arch|dcache|instr|tlbmem] [-t totalsize] mode

Modes (pick one - default 1thread):
    1thread                IPC within a single thread
    2thread                IPC between two threads in one process
    2proc                  IPC between two threads in two different processes

Optional flags:
    -B                     Run in bare mode: no preparatory activities
    -g                     Enable getrusage(2) collection
    -i pipe|local|tcp      Select pipe, local sockets, or TCP (default: pipe)
    -j                     Output as JSON
    -p tcp_port            Set TCP port number (default: 10141)
    -P arch|dcache|instr|tlbmem  Enable hardware performance counters
    -q                     Just run the benchmark, don't print stuff out
    -s                     Set send/receive socket-buffer sizes to buffersize
    -v                     Provide a verbose benchmark description
    -b buffersize          Specify the buffer size (default: 131072)
    -n iterations          Specify the number of times to run (default: 1)
    -t totalsize           Specify the total I/O size (default: 16777216)
```

- `-i tcp` Set IPC type to TCP
- `-p 10141` Set TCP port number

Important: There has been an update to the IPC benchmark to automatically flush the TCP host cache between iterations. **Please make sure that you are using the Lab 3 variant**

Loopback networking, IPFW, DUMMYNET

- Loopback network interface
  - Synthetic local network interface: packets “loop back” when sent
  - Interface name lo0
  - Assigned IPv4 address 127.0.0.1
  - **Set the MTU to 1500 bytes**

- IPFW – IP firewall by Rizzo, et al.
  - Numbered rules classify packets and perform actions
  - Actions include accept, reject, and inject into DUMMYNET
  - **Set up IPFW to match port 10141 and inject into DUMMYNET**

- DUMMYNET – Link simulation tool by Rizzo, et al.
  - Impose simulated network conditions (e.g., latency) on “pipes”
  - **Configure DUMMYNET pipes as required for the assignment**

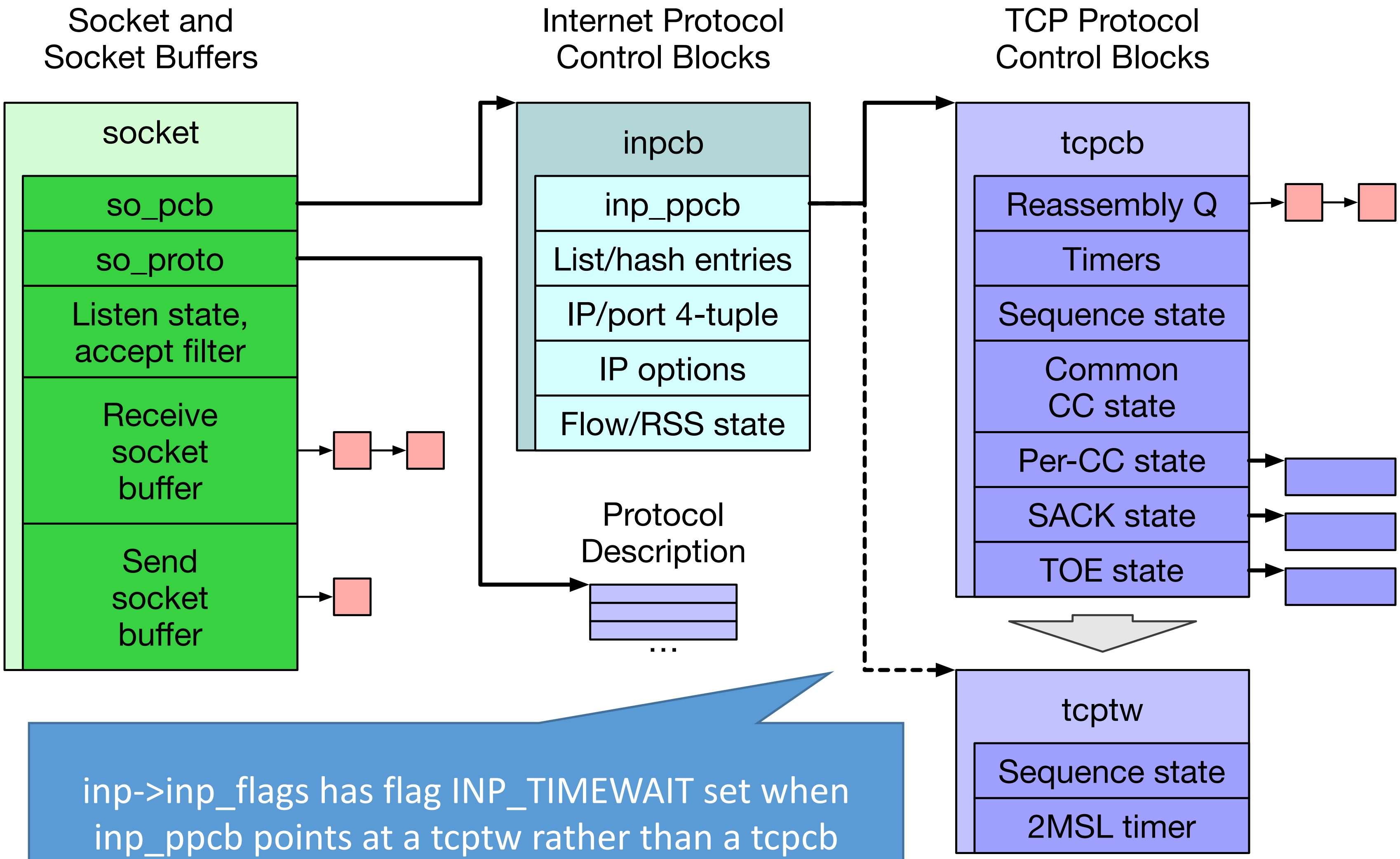
Some TCP-relevant DTrace probes

- Described in more detail in the lab assignment:

`fbt::syncache_add:entry`	TCP segment installs new SYN-cache entry
`fbt::syncache_expand:entry`	TCP segment converts SYN-cache entry to full connection
`fbt::tcp_do_segment:entry`	TCP segment received post-SYN cache
`fbt::tcp_state_change:entry`	TCP state transition

- We are using implementation-specific probes (FBT) rather than portable TCP provider probes in order to:
  - avoid the 5-argument limit to FreeBSD/arm64 DTrace; and
  - provide easier access to internal data structures
- Do not limit yourself to only these probes!

Lecture 6: Data structures – sockets, control blocks



Socket and Socket Buffers

socket

so_pcb

so_proto

Listen state, accept filter

Receive socket buffer

Send socket buffer

Internet Protocol Control Blocks

inpcb

inp_ppcb

List/hash entries

IP/port 4-tuple

IP options

Flow/RSS state

Protocol Description

…

TCP Protocol Control Blocks

tcpcb

Reassembly Q

Timers

Sequence state

Common CC state

Per-CC state

SACK state

TOE state

tcptw

Sequence state

2MSL timer

inp->inp_flags has flag INP_TIMEWAIT set when inp_ppcb points at a tcptw rather than a tcpcb

Part II: The TCP state machine

How does the TCP implementation state machine differ from the TCP protocol specification? How does latency affect transition through the state machine?

- Plot an effective (measured) TCP state-transition diagram for both directions of a flow
- Label the state-transition diagram with causes – TCP headers, system calls, timer, etc.
- Compare the diagram with RFC 793
- What observations can we make about state-machine transitions as latency increases?
- Describe any apparent simulation or probe effects

Part II: `tcpcb` sender-side data-structure fields

- In this lab, two parties have **`tcpcb`**s as we run:
  - The 'client' is receiving data
  - The 'server' is sending data

  } Instrument state transitions in both
- Described in more detail in the lab assignment:
  - **`t_state`** Current TCP state in a tcpcb
- Note that connection setup and teardown, there may not be a tcpcb present

L41: TCP congestion control

- This lab explores the behavior the TCP implementation and the bandwidth it achieves as latency is varied
    - How does TCP congestion control affect bandwidth at different latencies?
    - What are the impacts of specific implementation choices and policies, such as socket-buffer auto-sizing
- As we are working over the loopback interface, we can instrument both ends of the TCP connection
    - Track packet-level headers on transmit and receive
    - Also track TCP-internal parameters such as whether TCP is in “slow start” or the steady state
- And, of course, we care about the arising probe effect

L41: `tcpcb` sender-side data-structure fields

- In this lab, there are two parties with **`tcpcb`**s as we run:
  - The ‘client’ is receiving data
  - The ‘server’ is sending data ← **Instrument CC send state here**
- For the purposes of classical TCP congestion control, only the sender retains congestion-control state
- Described in more detail in the lab assignment:

  `snd_wnd` Last received advertised flow-control window.

  `snd_cwnd` Current calculated congestion-control window.

  `snd_ssthresh` Current slow-start threshold:

  if (`snd_cwnd <=` `snd_ssthresh`), then TCP is in slowstart; otherwise, it is in congestion avoidance
- Instrument **`tcp_do_segment`** using DTrace to inspect TCP header fields and **`tcpcb`** state for **only the server**
  - Inspect port number to decide which way the packet is going

L41: Experimental questions for the lab report

1. How do latency and achieved TCP bandwidth relate?
    - Plot DUMMYNET-imposed latency on the X axis and effective bandwidth on the Y axis, considering both the case where the socket-buffer size is set versus allowing it to be auto-resized.
2. How does socket-buffer strategy interact with latency?
    - Plot a time-bandwidth graph comparing the effects of setting the socket-buffer size versus allowing it to be auto-resized by the stack. Stack additional graphs showing the sender last received advertised window and congestion window on the same X axis.
3. Be sure to describe any simulation or probe effects.

Get in touch if you need a hand

- You can reach me and the course demonstrators on Slack – we try to reply quickly
- We will arrange 1:1 supervision sessions at various points during the assignment period
  - Read the assignment, experiment a bit first
  - Talk to us about the strategies you are pursuing – or if you aren’t sure what strategy to pursue
  - Ask us if you have questions about what you discover
- Or drop me email directly