L41: Lab 4 - The TCP State Machine

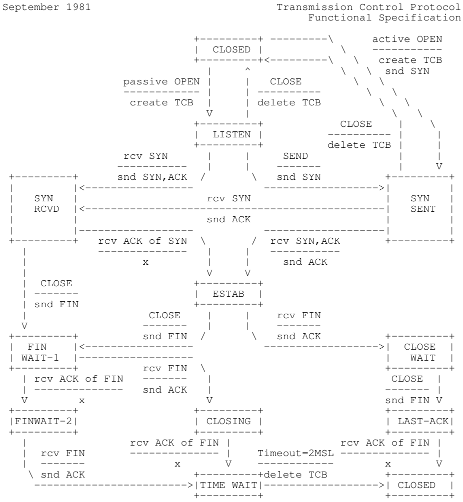
Dr Robert N. M. Watson Dr Graeme Jenkinson

3 February 2017

L41: Lab 4 - The TCP State Machine

- The TCP state machine
- Setting the MTU, IPFW, and DUMMYNET
- TCP mode for the IPC benchmark
- DTrace probes of interest
- Plotting the state machine with Graphviz
- Experimental questions

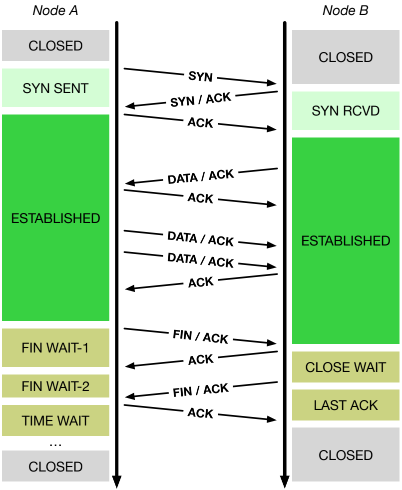
Lect 6: The Transmission Control Protocol (TCP)



TCP Connection State Diagram
Figure 6.

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

Lect 6: TCP goals and properties



- Network may delay, (reorder), drop, corrupt packets
- TCP: Reliable, ordered, stream transport protocol over IP
- Three-way handshake: SYN / SYN-ACK / ACK (mostly!)
- Sequence numbers ACK'd; data retransmitted on loss
- Round-Trip Time (RTT) measured to time out loss
- Flow control via advertised window size in ACKs
- Congestion control ('fairness') via packet loss and ECN

Loopback interface, IPFW, and DUMMYNET

- Network-stack features to configure **once per boot**
- Loopback interface
  - Simulated local network interface: packets “loop back”
  - Interface name `lo0`
  - Assigned IPv4 address `127.0.0.1`
- IPFW - IP firewall by Rizzo, et al.
  - Numbered rules classify packets and perform actions
  - Actions include accept, reject, inject into DUMMYNET ...
  - We will match lab flows using the TCP port number `10141`

- Configure (and reconfigure) **for each experiment**
- DUMMYNET - link simulation tool by Rizzo, et al.
  - Widely used in network research
  - Impose simulated network conditions – delay, bandwidth, loss, ...

TCP in the IPC benchmark

```
root@beaglebone:/data/ipc # ./ipc-static
ipc-static [-Bqsv] [-b buffersize] [-i pipe|local|tcp] [-p tcp_port]
        [-P l1d|l1i|l2|mem|tlb|axi] [-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
    -i pipe|local|tcp      Select pipe, local sockets, or TCP (default: pipe)
    -p tcp_port            Set TCP port number (default: 10141)
    -P l1d|l1i|l2|mem|tlb|axi  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 a buffer size (default: 131072)
    -t totalsize           Specify total I/O size (default: 16777216)
```

- `tcp` IPC type
- `-p` argument to set the port number

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 probes due to a bug in the FreeBSD/armv7 implementation of DTrace – the last (and most critical!) argument goes missing: the TCP header! We will fix this .. but not today.

Graphviz

Graphviz is open source graph visualization software for drawing graphs specified in DOT language scripts. This language describes three kinds of objects: graphs, nodes, and edges.

- **Graph** `graph` or `digraph` undirected or directed graph
- **Node** `syn-sent;`
- **Edge** `"closed" -> "syn-sent";`

Nodes and Edges can be assigned attributes changing, for example, their colour or shape:

- **Node** `syn-sent [color=blue];`
- **Edge** `"closed" -> "syn-sent" [label="Active open", color=green];`

pygraphviz

Programmatic interface for creating visualizations with Graphviz.

```
>>> import pygraphviz as pgv
>>> G = pgv.AGraph(strict=False, directed=True)
>>> G.add_node('a')
>>> G.add_edge('b','c')
>>> print(G)
digraph {
    a;
    b -> c;
}
```

pygraphviz graphs can be viewed directly in a Juyter Notebook (see laboratory template).

Experimental questions for the lab report

- Plot a TCP state-transition diagram for both directions of a flow
- Label the state-transition diagram with causes
- Compare the diagram with RFC 793
- What observations can we make about state-machine transitions as latency increases?

In the next lab, we will start a causal analysis of why latency affects bandwidth in the way that it does

This lab session

- Set up IPFW, DUMMYNET, and loopback MTU (see notes)
- Ask us if you have any questions or need help
- Start with the TCP state machine analysis