Hybrid Start
A TCP Slow Start Algorithm

Narayan Pai
Final year CSE,
NITK Surathkal

narayanspai2000@gmail.com
Roll No: 181CO135

Outline

1. Congestion Control
2. Phases in Congestion Control Algorithm
3. Problems with standard slow start
4. Hybrid Start Algorithm
5. Kernel Code Walkthrough
6. Experimenting with the kernel

Congestion Control

- ➔ It is a mechanism within TCP that makes sure congestion does not occur in the network.
- ➔ Enforced with a state variable named Congestion Window (***cwnd***) that limits the number of packets the sender can send into the network before receiving an ACK.

Phases in Congestion Control

Slow Start Phase

Starting with a small value, increment cwnd exponentially to ssthresh

Congestion Avoidance Phase

Increment cwnd linearly

Congestion Detection Phase

Decrement cwnd and ssthresh.

**ssthresh (slow start threshold) is a dynamic threshold for cwnd*

Phases in Congestion Control Algorithm

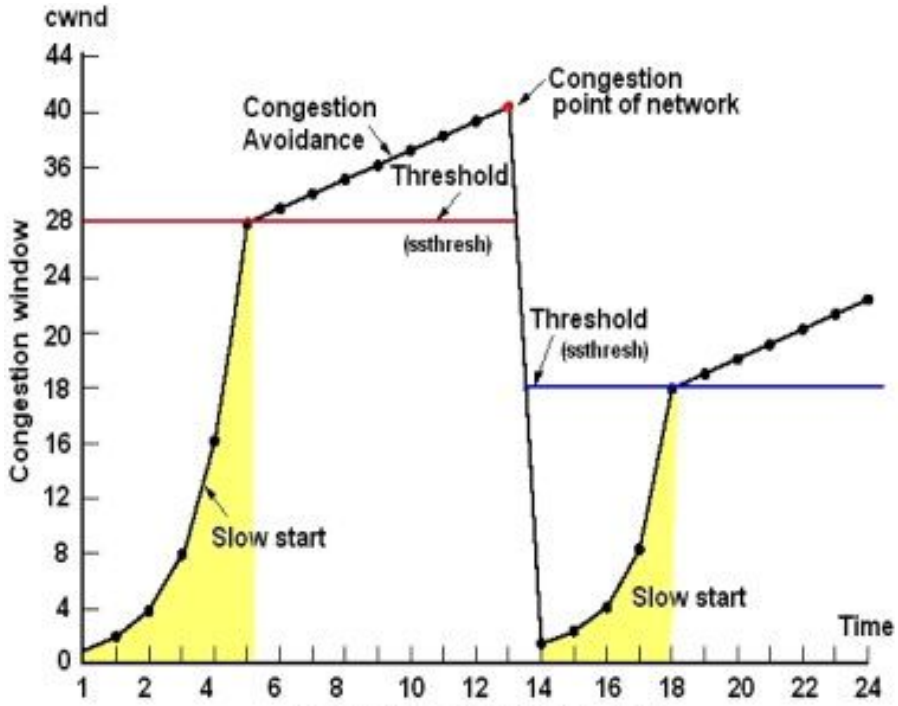


Image Credits: https://www.researchgate.net/

Problems with standard slow start

1. **Heavy packet losses** occur because of the exponential increase of the congestion window during standard slow start.
2. Results in a **long blackout period** without transmission.

For eg, second packet is dropped when cwnd = 100.

Activity: Think about what happens if there are other packet drops too?

Hybrid Start : The solution?

Introduced in 2009 by Sangtae Ha and Injong Rhee in a paper named *"**Taming the elephants: New TCP slow start**"*

➔ **Uses**

ACK trains and RTT delay samples

➔ **To**

Heuristically finds safe exit points at which it can finish slow start and move to congestion avoidance before cwnd overshoots

Fact

It is used in TCP Cubic, the default TCP in Linux.

Packet train

Set of packets of same length sent back to back from the sender to the receiver to probe the network condition.

If N packets of length L are sent, then the receiver can estimate bandwidth by:

$$b(N) = \frac{(N-1)L}{\Delta(N)}$$

Δ(N) = Packet train length
= Inter-arrival time between first and the last packet

Bandwidth Delay Product

- Bandwidth delay product is a measurement of how many bits can fill up a network link.

- Gives an upper bound for cwnd

Bandwidth Delay Product

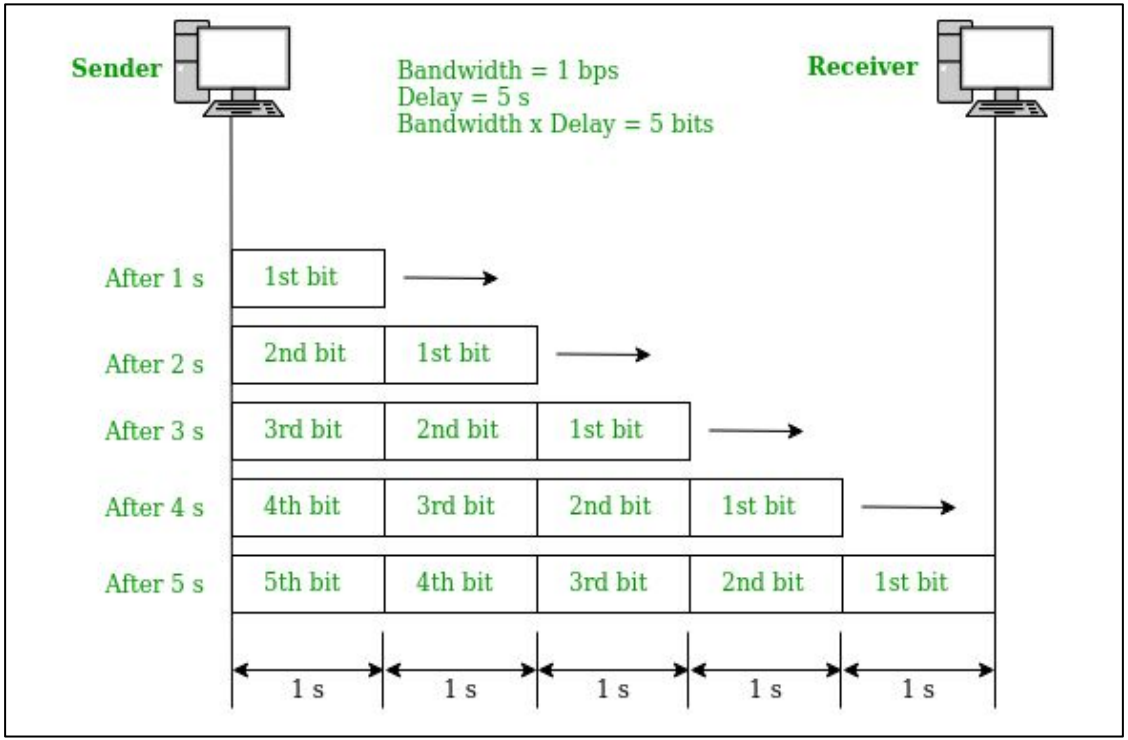


Image Credits: https://www.geeksforgeeks.org/

Exit Criterion #1: ACK Train length

Hystart uses the data packets itself as probe packet train.

$$cwnd.L = (N-1).L \quad \text{- (1)}$$

During complete network utilization,

$$cwnd.L = b(N) \cdot D_{min} = \frac{(N-1).L}{\Delta(N)}.D_{min} \qquad \text{- - (2)}$$

By (1) and (2), we get $\Delta(N) = D_{min}$ during complete network utilization.

Conclusion: Exit when $\Delta(N) > D_{min}$

Implementation Details

1. To estimate Δ(N), we use the train of ACKs received in response to a packet train. Helps in conservative estimation of the available capacity of the path.
2. To estimate D_{min}, we divide the minimum RTT by two.

Problems!

1. When other flows are competing for the same path, D_{min} doesn't give us the required value.
2. Not effective when the network is asymmetric.

Exit Criterion #2: Delay increase

Step 1: Measure minimum RTT of a sample in each round.

Step 2: If it is considerably greater than that of the previous round, exit.

Linux Code Walkthrough

Configuration variables

net/ipv4/tcp_cubic.c

```
/* Two methods of hybrid slow start */
#define HYSTART_ACK_TRAIN 0x1
#define HYSTART_DELAY   0x2
/* Number of delay samples for detecting the increase of delay */
#define HYSTART_MIN_SAMPLES 8
#define HYSTART_DELAY_MIN (4000U) /* 4 ms */
#define HYSTART_DELAY_MAX (16000U)  /* 16 ms */
#define HYSTART_DELAY_THRESH(x) clamp(x, HYSTART_DELAY_MIN, HYSTART_DELAY_MAX)

static int hystart __read_mostly = 1;
static int hystart_detect __read_mostly = HYSTART_ACK_TRAIN | HYSTART_DELAY;
static int hystart_low_window __read_mostly = 16;
```

On receiving each ACK

net/ipv4/tcp_cubic.c

```
static void hystart_update(struct sock *sk, u32 delay)
{
        struct tcp_sock *tp = tcp_sk(sk);
        struct bictcp *ca = inet_csk_ca(sk);
        u32 threshold;

        if (hystart_detect & HYSTART_ACK_TRAIN) {
            u32 now = bictcp_clock_us(sk);

            ca->last_ack = now;
            threshold = ca->delay_min + hystart_ack_delay(sk);

            if ((s32)(now - ca->round_start) > threshold) {
              ca->found = 1;
              NET_INC_STATS(sock_net(sk),
                            LINUX_MIB_TCPHYSTARTTRAINDETECT);
              NET_ADD_STATS(sock_net(sk),
                            LINUX_MIB_TCPHYSTARTTRAINCWND,
                            tp->snd_cwnd);
              tp->snd_ssthresh = tp->snd_cwnd;
            }
        }
```

contd ...

On receiving each ACK (contd.)

net/ipv4/tcp_cubic.c

```
        if (hystart_detect & HYSTART_DELAY) {
            /* obtain the minimum delay of more than sampling packets */
            if (ca->curr_rtt > delay)
              ca->curr_rtt = delay;
            if (ca->sample_cnt < HYSTART_MIN_SAMPLES) {
              ca->sample_cnt++;
            } else {
              if (ca->curr_rtt > ca->delay_min +
                  HYSTART_DELAY_THRESH(ca->delay_min >> 3)) {
                ca->found = 1;
                NET_INC_STATS(sock_net(sk),
                        LINUX_MIB_TCPHYSTARTDELAYDETECT);
                NET_ADD_STATS(sock_net(sk),
                        LINUX_MIB_TCPHYSTARTDELAYCWND,
                        tp->snd_cwnd);
                tp->snd_ssthresh = tp->snd_cwnd;
              }
            }
         }
}
```

Experimenting with the kernel

Tools Used: Flent, NeST

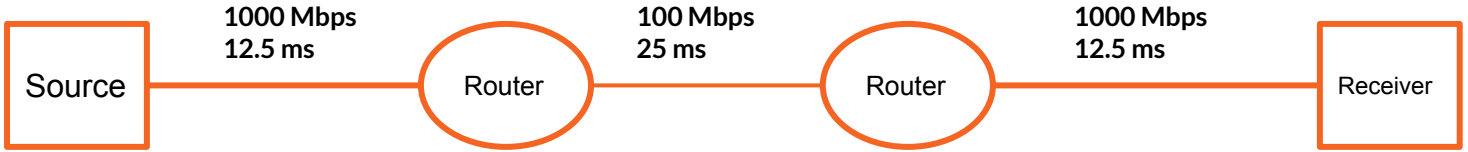
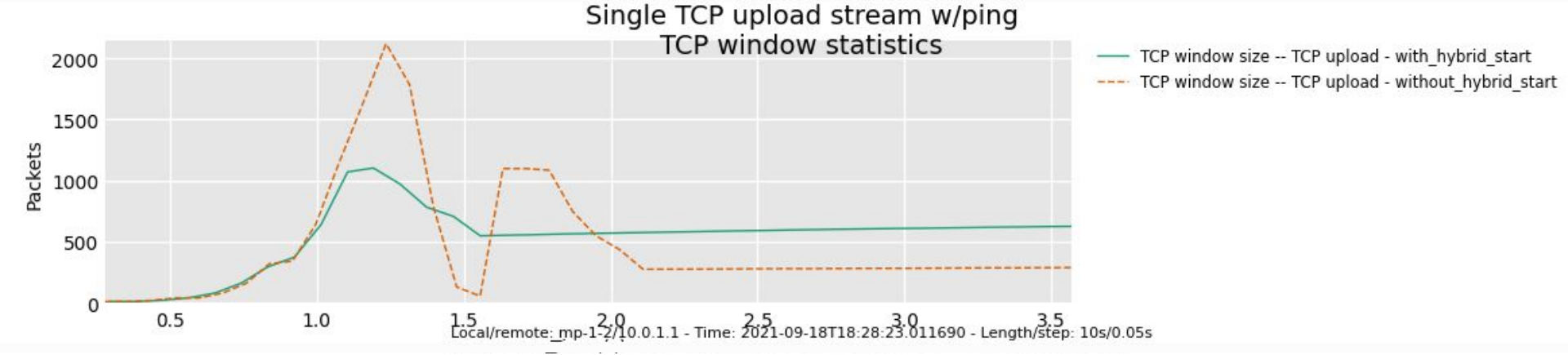


Fig: Experiment Topology with link bandwidth and delay

Step 1: Run tests with default kernel
Step 2: Download the kernel source
Step 3: Customize the kernel
Step 4: Build and install the kernel
Step 5: Run tests with the customized kernel

Results


Single TCP upload stream w/ping
TCP window statistics
Packets
2000
1500
1000
500
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
TCP window size -- TCP upload - with_hybrid_start
TCP window size -- TCP upload - without_hybrid_start
Local/remote: mp-1-2/10.0.1.1 - Time: 2021-09-18T18:28:23.011690 - Length/step: 10s/0.05s

Bibliography

1. *“Taming the elephants: New TCP slow start”*: https://doi.org/10.1016/j.comnet.2011.01.014
2. Instructions, code and results of the experiment: https://github.com/narayanpai1/hybrid-start-experiments
3. Flent tool: https://flent.org
4. NeST tool: http://nest.nitk.ac.in

Thank You!