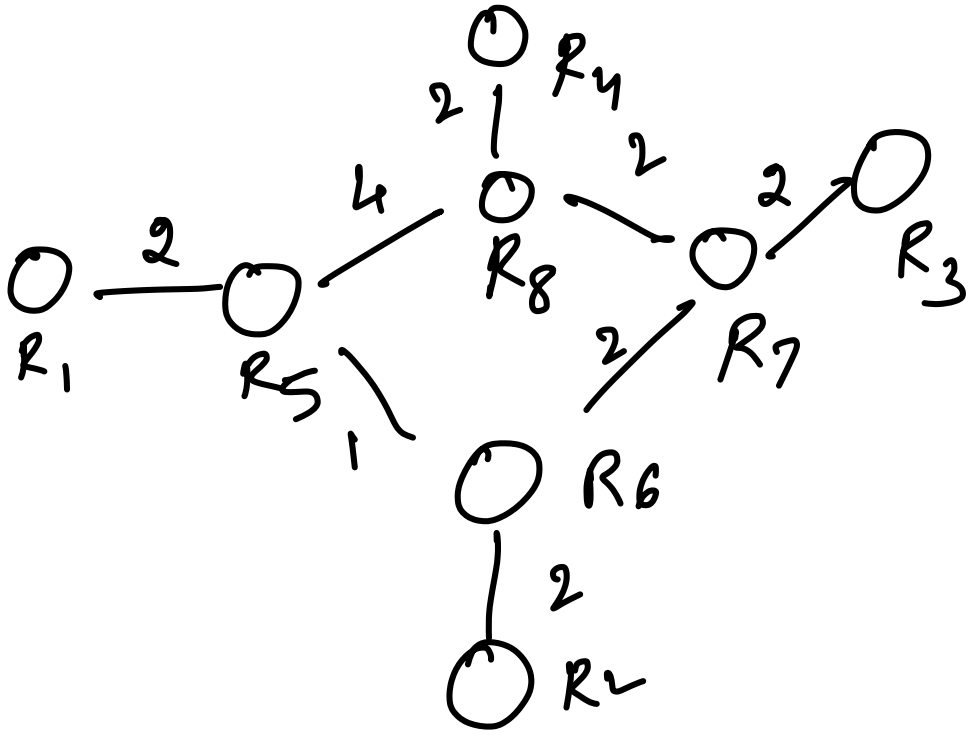
CS 224 Assignment 4

Q1)

(a)



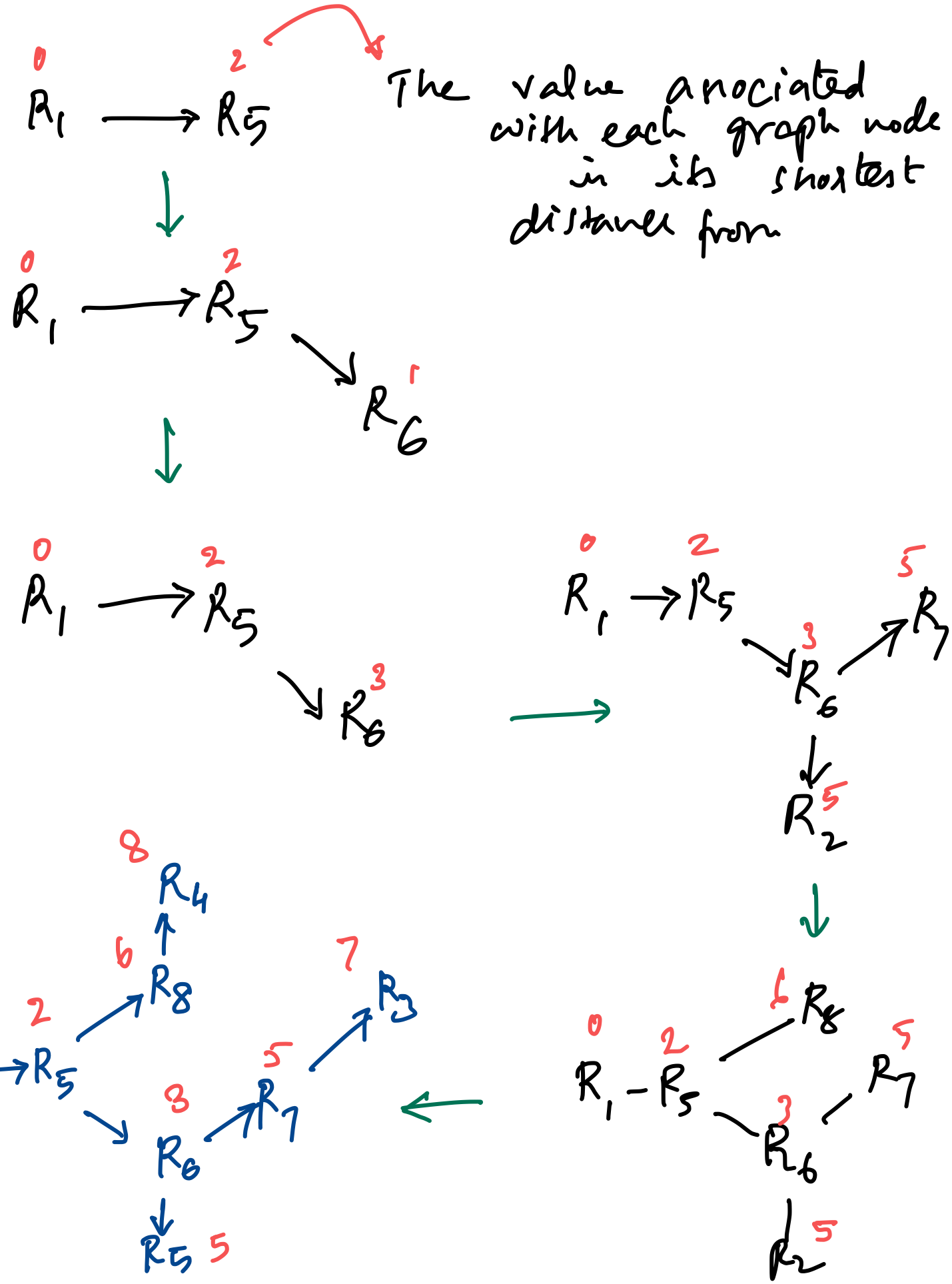
using Dijkstra's algorithm to find shortest path from R1 to all other routers

all nodes give information about their immediate neighbour's linkspeeds

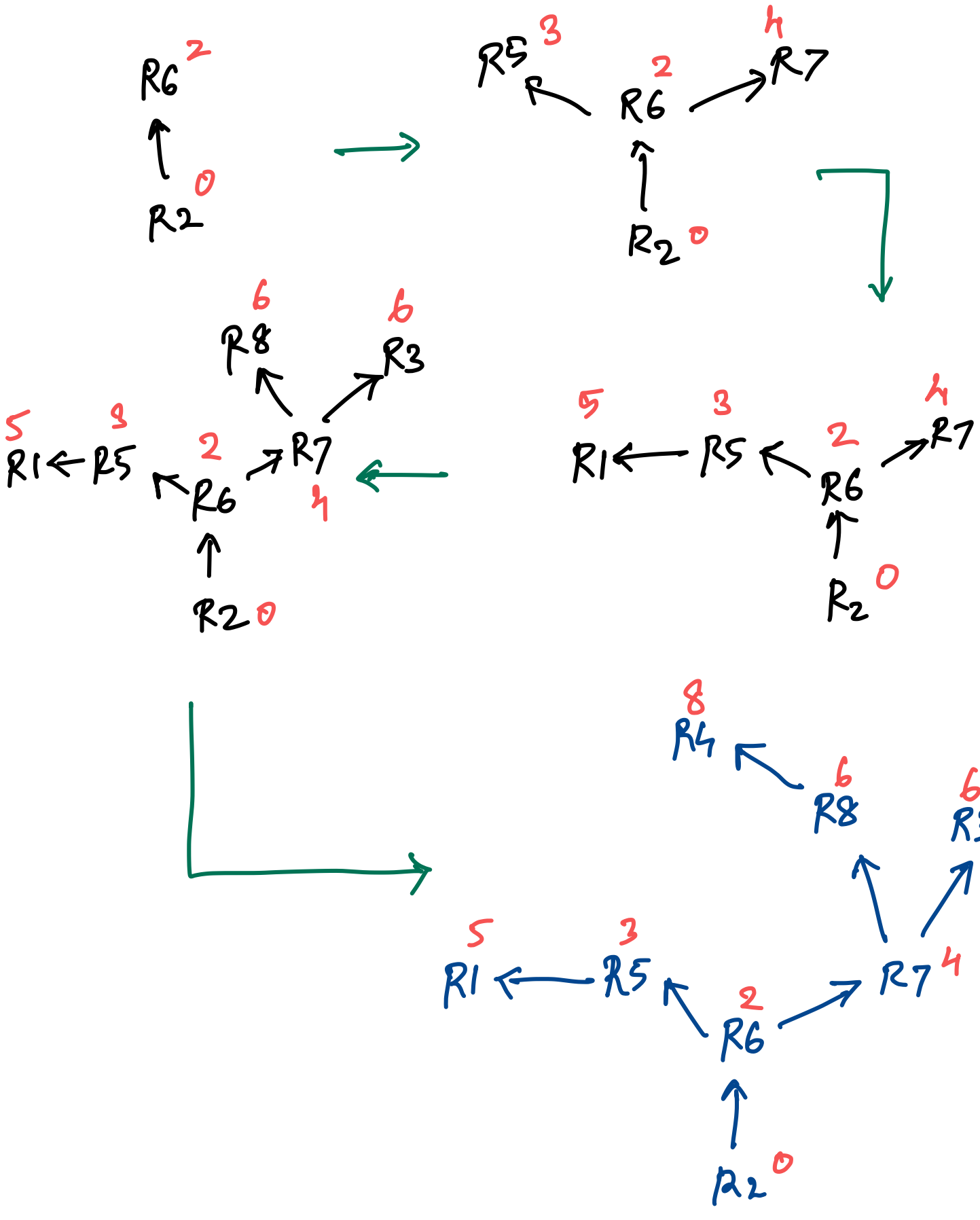
we then choose among the closest nodes and expand the routing tree.

we can use Dijkstra as the link weights are all positive.

Creation of Routing Tree



similarly for the router R2



(b) given advertisements are

[130.12.1.0/24 AS3]

[142.13.0.0/16 AS3 -AS4]

[151.128.32.0/24 AS3 - AS4 - AS2]

from R13 to R3.

[151.128.32.0/24 AS2]

[142.13.0.0/16 AS2-AS4]

[130.12.1.0/24 AS2 - AS4 - AS3]

from R14 to R4.

given LOCAL-PREF is same and no MED

(i) all BGP routers choose R14 as NEXT_HOP router

(a) [151.128.32.0/24 AS3 - AS4 - AS2] from R13

(b) [151.128.32.0/24 AS2] from R14

as out of these two both have same LOCAL_PREF, but out of these (b) has shortest AS_PATH among (a) and (b).

(ii) all BGP routers choose R13 as NEXT_HOP router

(a) [130.12.1.0/24 AS3] from R13

(b) [130.12.1.0/24 AS2 - AS4 - AS3] from R14

as out of these two both have same LOCAL_PREF, but out of these (a) has shortest AS_PATH among (a) and (b).

(iii)

(a) [142.13.0.0/16 AS3 - AS4] from R13

(b) [142.13.0.0/16 AS2 - AS4] from R14

since both (a), (b) have same LOCAL_PREF, same AS_PATH length, no MED.

for BGP router R4

→ R14 in NEXT_HOP router

as (b) is learnt over eBGP and (a) is learnt over iBGP.

for BGP router R3

→ R13 in NEXT_HOP route as (a) is learnt over eBGP and (b) is learnt over iBGP.

for BGP route R2

(a), (b) both are learnt over iBGP so we need to use Hot Potato routing

→ R13 in NEXT_HOP route as by Hot Potato Routing we get dist(R2, R3) = 6

dist(R2, R4) = 8

dist(R2, R3) < dist(R2, R4).

for BGP route R1

(a), (b) both are learnt over iBGP so we need to use Hot Potato routing

→ R13 in NEXT_HOP route as dist(R1, R3) = 7 < dist(R1, R4) = 8.

(iv) if admin (AS1) wants all BGP routers to use NEXT_HOP as R14 for packets addressed to 142.13.0.0/16

(a) [142.13.0.0/16 AS3 - AS4] from R13

(b) [142.13.0.0/16 AS2 - AS4] from R14

1. he can set LOCAL_PREF for (b) to be higher than (a)

2. he can set lower MED for (b) than (a) in case LOCAL_PREF is not modifiable.

(c) Packet P1 is forwarded to R1 from AS5 with dest 142.13.5.4

also 142.13.5.4 ∈ 142.13.0.0/16

so R1 decides by BGP rules to set NEXT_HOP router as R13

as explained in previous section.

(i) R1 encapsulate P1, R3 eventually de-encapsulates it.

The new packet formed after encapsulation @ R1 is given by P2

The dest IP corresponds to the IP address of router R3.

(ii) Only the IGP routing tables are looked up by the various routers to find the next hop to forward P2 as P1 is completely encapsulated in P2.

The next router in the shortest path from R1 — R3 closest to the current router should be in the next hop of the table.

for example) at R6 next hop should be R7.

(Q2) The design for new TCP variant

Assumptions

↳ we assume that all flows have the same value of RTT.

↳ There is no slow start (it is mentioned in slow start)

↳ All vegas flows have same value of α, β.

New variant TCP rules

- For the flow we set the value of α, β same as other vegas.
- we keep track of instantaneous RTT as $r[t]$ (a discrete sequence)
- we evaluate instantaneous $diff\{d[t]\}$ as in TCP vegas

$$d[t] = W\left[\underbrace{\frac{1}{\min_t\{r[t]\}}}_{\text{base RTT}} - \frac{1}{r[t]}\right]$$

- define a new threshold $\hat{\beta} > \beta$ for checking presence of Reno flows

- if $(d[t] < \alpha)$ then perform an additive increase of cw as $cw = cw + \frac{(1\ MSS)^2}{cw}$.
- if $(d[t] >= \beta)$ then perform an additive decrease of cw as $cw = cw - \frac{(1\ MSS)^2}{cw}$ and set a timeout of $\hat{T}$ iterations where $d[t+n]$ in not compared with $\hat{\beta}$, $1 \leq n \leq \hat{T}$
- if $(\alpha \leq d[t] \leq \beta)$ then keep the congestion window size same

{ Note: All TCP vegas flows in this particular link will perform additive decrease and $d[t+\hat{T}]$ will stay less than $\hat{\beta}$.
However if there is atleast 1 TCP Reno link in this, it will still perform additive increase and $d[t+\hat{T}] > \hat{\beta}$ }

- Compare the value of $d[t+\hat{T}]$ with $\hat{\beta}$, if it is greater, then set the value of α, β to a very large value (inf or INT_MAX) such that it behaves like TCP Reno

- If $d[t+\hat{T}]$ is lesser than $\hat{\beta}$, then all flows are TCP Vegas, hence we leave α, β as it is and resume additive increase

- if 3 DUP ACKs are got (congestion) set $cw = \frac{cw}{2}$ and α, β = INT_MAX as queue is filled means a Reno flow exists.

- If a network timeout happens set cw = MSS and restart the process of additive increase and reset α, β to initial values

- Note: The values of $\hat{\beta}$ and $\hat{\tau}$ must be chosen according to the capacity of the link and the queue buffer experimentally

Pseudo Code

```
while (true) {
  r[t] = getrtt(); // gets current RTT
  br = min(r[t], br);
  d[t] = cw(1/br - 1/r[t]);
  if (d[t] < α)
      cw = cw + (1 MSS)^2/cw;
  else if (α < d[t] < β)
      cw = cw;
  else{
      d[t+τ̂] = check(τ̂);
      if (d(t+τ̂) > β̂ ){
          α = INT_MAX; β = INT_MAX;
      }
      t += τ̂;
      continue;
  }
}
```

```
int check(T̂) {
 int i=1; int K;
    while (i<=T̂) {
      r[t+i]= getrtt();
       K = cw( 1/br - 1/r[t+i] );
      cw = cw - (1MSS)²/cw;
       i++;
    }
   return K;
}
```

It satisfies all properties a, b, c, d as shown below.

(a) It is an end-to-end TCP protocol as it doesn't need any sort of information from other TCP flows on routers. It works purely on link RTT, packet losses.

(b) As shown earlier it behaves like TCP vegas and keeps α, β unchanged if all other flows are TCP vegas which happens iff $d[t+\hat{T}] \lesssim \hat{\beta}$

(c) As shown earlier if $d[t+\hat{T}]$ is greater than $\hat{\beta}$, then in that case α, β is set to be INT_MAX and then the flow behaves like TCP Reno.

(d) The adjustments of CW happen out of inferences of the value of $d[t]$, which depends on queueing delay as well as packet losses only.

(Q3) Both flows face "loss" at time t if

$$\frac{\omega_1(t)}{T_1} + \frac{\omega_2(t)}{T_2} = C$$

after which $\omega_1(t)$ and $\omega_2(t)$ are both set to 0.

$$\frac{d\omega_1(t)}{dt} = \frac{1}{T_1} \qquad \frac{d\omega_2(t)}{dt} = \frac{1}{T_2}$$

$$\omega_1(t) = t/T_1 \qquad \omega_2(t) = t/T_2$$

as $\omega_i(0) = 0$ for both.

since we are ignoring slow start, time $\hat{t}$ @ which both $\omega_1(t)$ and $\omega_2(t)$ are set to 0 is

$$\frac{\hat{t}}{T_1^2} + \frac{\hat{t}}{T_2^2} = C \qquad \hat{t}\left(\frac{1}{T_1^2} + \frac{1}{T_2^2}\right) = C$$

$$\hat{t} = \frac{C}{\left(\frac{1}{T_1^2} + \frac{1}{T_2^2}\right)} = \frac{C T_1^2 T_2^2}{T_1^2 + T_2^2}$$

we wish to evaluate data transmitted by the two flows b/w to consecutive "loss" events

suppose we had a "loss" event at $t=0$ and $w_1(t)$ and $w_2(t)$ are both set to 0.

instantaneous bitrate for flow i

$$\frac{dB_i}{dt} = \frac{w_i(t)}{T_i} \cdot \quad B_i(t) = \int \frac{w_i(t)}{T_i} dt$$

amount of bits transmitted by flow 1 is

$$\int_0^{\hat{t}} \frac{w_1(t)}{T_1} dt = \int_0^{\hat{t}} \frac{t}{T_1^2} \cdot dt = \frac{\hat{t}^2}{2T_1^2}$$

$$= \frac{C^2 T_1^4 \cdot T_2^4}{(CT_1^2 + T_2^2)^2} \times \frac{1}{2T_1^2}$$

$$= \frac{C^2 T_1^2 \cdot T_2^4}{(CT_1^2 + T_2^2)^2} \text{ bits}$$

amount of bits transmitted by flow 2 in $\hat{t}$

$$\int_0^{\hat{t}} \frac{w_2(t)\,dt}{T_2} = \int_0^{\hat{t}} \frac{t}{T_2^2}\cdot dt = \frac{\hat{t}^2}{2T_2^2}$$

$$= \frac{C^2 \cdot T_1^4 \cdot T_2^4}{(T_1^2+T_2^2)^2} \cdot \frac{1}{2T_2^2}$$

$$= \frac{C^2 T_1^4 T_2^2}{(T_1^2+T_2^2)^2} \text{ bits}$$

Total data transmitted

$$\frac{C^2 T_1^4 T_2^2}{(T_1^2+T_2^2)^2} + \frac{C^2 T_1^2 T_2^4}{(T_1^2+T_2^2)^2} = \frac{C^2 T_1^2 T_2^2}{(T_1^2+T_2^2)}$$

fraction corresponding to flow 1

$$= \frac{C^2 T_1^2 T_2^4}{(T_1^2+T_2^2)^2} \cdot \frac{(T_1^2+T_2^2)}{C^2 T_1^2 \cdot T_2^2} = \frac{T_2^2}{T_1^2+T_2^2}$$

fraction corresponding to flow 2

$$= 1 - \frac{T_2^2}{T_1^2 + T_2^2} = \frac{T_1^2}{T_1^2 + T_2^2}.$$

Also the amount of data of a flow depends on the RTT as follows

$$B_i(t) \propto \frac{T_i^4}{(C + T_i^2)^2}$$

where C is a constant.

TCP Reno is fair if T_1 and T_2 are comparable, because in that case both $\frac{T_1^2}{T_1^2 + T_2^2}$ and $\frac{T_2^2}{T_1^2 + T_2^2} \simeq 0.5$.

However if T_1^2 is very large compared to T_2^2 or vise versa, in that case TCP Reno favours the flow with lower value of RTT.