RCS: A Rate Control Scheme for Real-Time Traffic in Networks with High B X Delay and High error rates

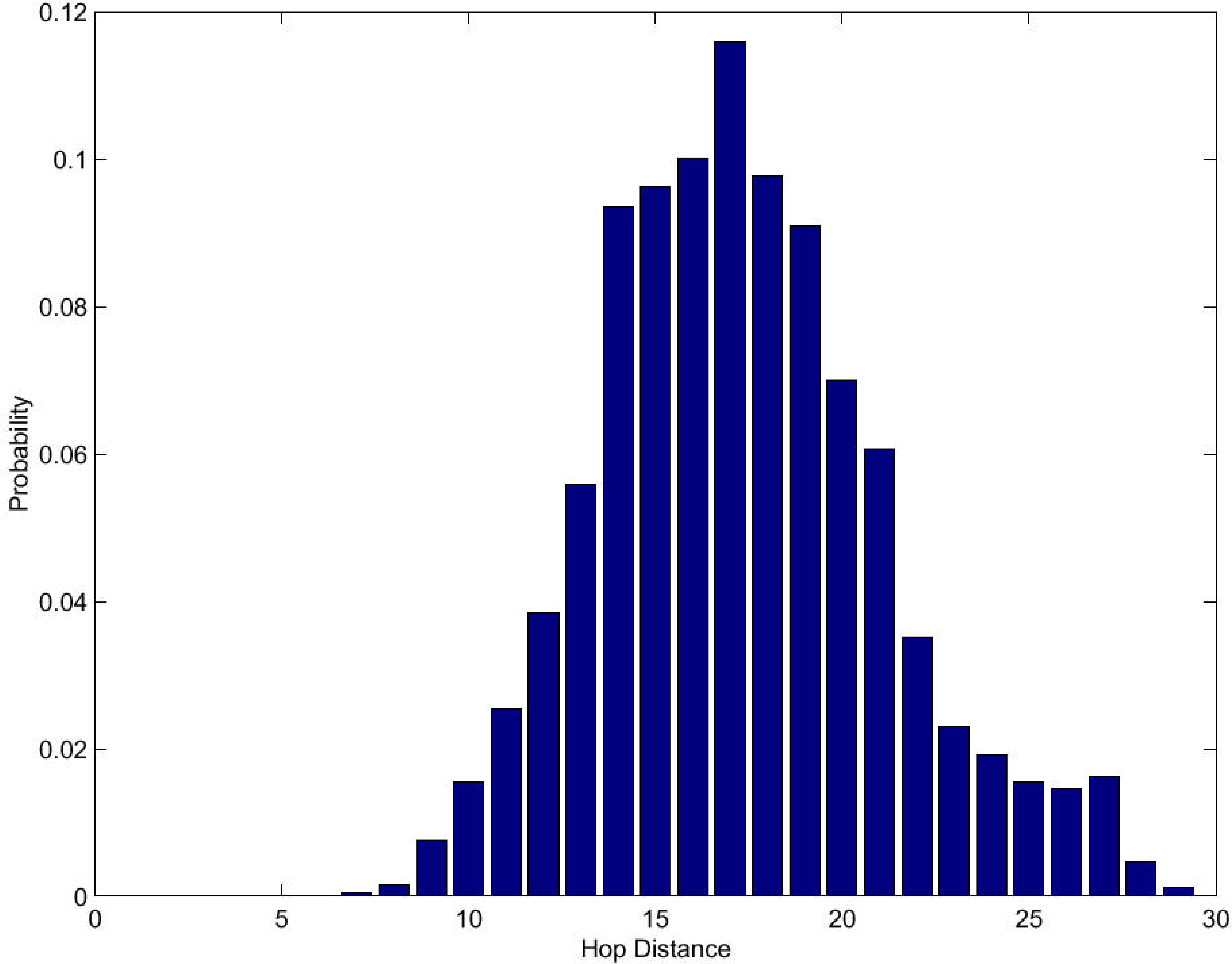
J. Tang et al , Infocom 2001

- Another streaming control protocol
- Application level
- Assumes fine grain layered encoding
- Targets the channels with **loss due to errors**
- TCP friendliness is secondary (but also important) concern

Large B X Delay situation

- Delay are growing higher in the Internet
- Avg hop distance is 16

University	Country	*RTT*
Georgia Tech	USA	200 msec
University of Campinas	Brasil	420 msec
Korea University	Korea	430 msec
Beijing University	China	800 msec


0.12
0.1
0.08
0.06
0.04
0.02
0
Probability
0
5
10
15
20
25
30
Hop Distance

Problems with wireless lossy links

- Conventional TCP cannot distinguish between errors and buffer O/F
- Some wireless links (eg satellites) have high packet loss rate (> .01)
- TCP efficiency drops to less than 20%!
- We need to improve TCP as well as TCP friendly streaming protocols for such lossy environments

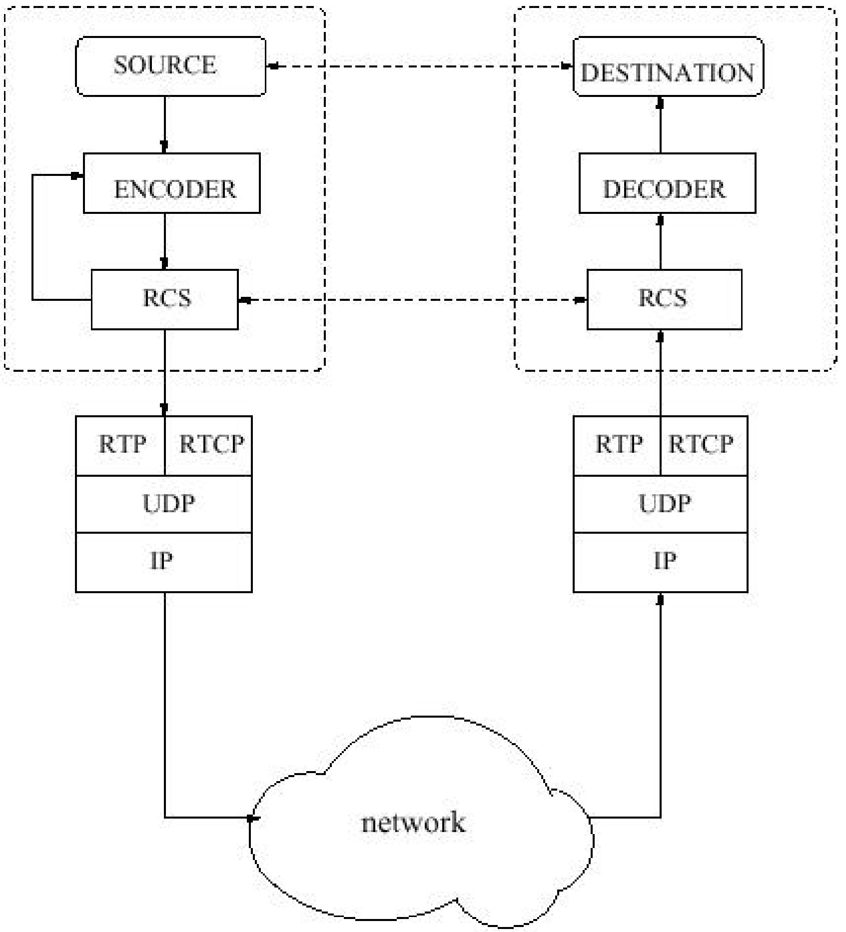
RCS: the goals

- RCS (Rate Control Scheme) is a TCP friendly rate control scheme for streaming
- It is robust to link errors
- It performs like TCP in error free situations
- It outperforms TCP in high error environments (without penalizing TCP)

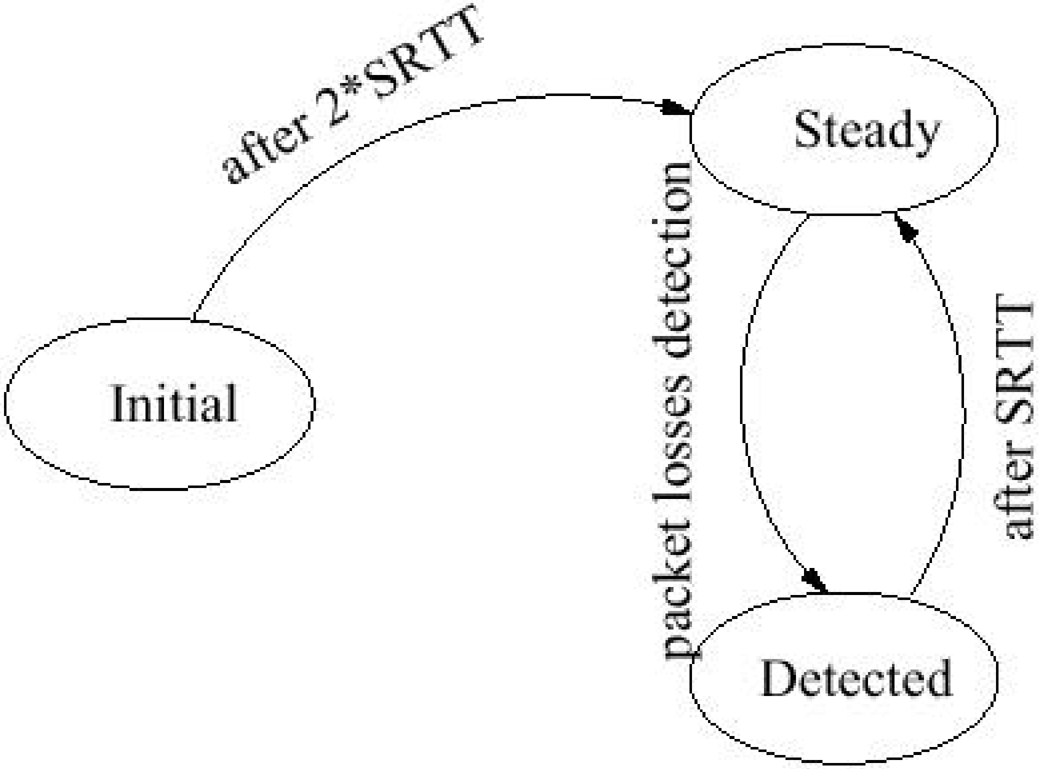
RCS: key ideas

- Source **probes** the connection with *dummy* packets
- Congested router drops *dummy* pkts first (lowest priority)
- Surviving *dummies* are ACKed by destination
- Source uses *dummy feedback* to increase and decrease rate

RCS: “middleware” level implementation



RCS: state transition diagram



Initializion phase

- Source probes the connection for available resources with dummy pkts
- Dummy pkt rate = S-target rate
- Let us say, n dummy pkts are ACK'd
- Initial rate = n/SRTT
- SRTT is the RTT measured by the source

```
Initial()
    t_0 = t;
    t_1 = t_0 + SRTT;
    t_END = t + 2 · SRTT;
    IPG_Dummy = 1/S_Target;
    t_next_dummy = t_0 + IPG_Dummy;
    n_ACK = 0;
    while (t ≤ t_END)
        while (t ≤ t_1)
            while(t < t_next_dummy)
                if (DUMMY_ACK_ARRIVAL)
                    n_ACK = n_ACK + 1;
                end;
            end;
            send(DUMMY_PACKET);
            t_next_dummy = t_next_dummy + IPG_Dummy;
        end;
        if (DUMMY_ACK_ARRIVAL)
            n_ACK = n_ACK + 1;
        end;
    end;
    wdsn = −1;
    S = max(1, n_ACK)/SRTT;
    state=Steady;
end.
```

```
Steady()
    END=0;
    t_0 = t;
    t_{next_data} = t_0;
    t_{next_increase} = t_0 + SRTT;
    while (END == 0)
        if (PACKET_LOSS_DETECTION)
            END=1;
        end;
        if (t ≥ t_{next_data})
            send(DATA_PACKET);
            t_{next_data} = t_{next_data} + IPG;
        end;
        if (t ≥ t_{next_increase})
            S = min(S + 1/SRTT, S_{Target});
            IPG = 1/S;
        if (DUMMY_ACK_ARRIVAL)
            if (wdsn == 0)
                S = min(S + 1/SRTT, S_{Target});
                IPG = 1/S;
            else
                wdsn = wdsn - 1;
            end;
        end;
    end;
    state=Detected;
end.
```

RCS: steady state behavior

- In steady state behavior (no errors detected) the sender increases the rate by one packet per SRTT after each SRTT cycle
- Rate increase is stopped when S-target is reached

RCS: detected loss state

- Sender cuts rate by half when it **detects loss** (the receiver explicitly informs sender of loss via dup ACKs as in RAP; or NACKs)
- The sender also probes (for a SRTT interval) the path with dummy pkts (two *dummies* for each data pkt) => rate =3/2 S
- After SRTT, sender returns to steady state and monitors the return of *dummy* ACKs

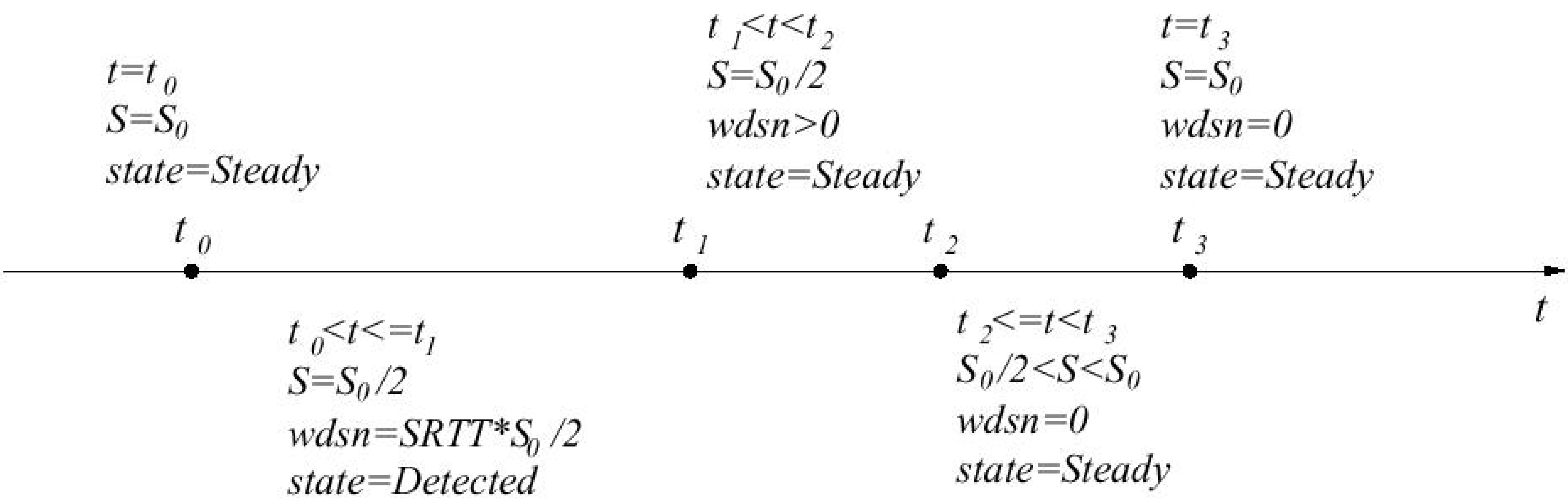
RCS: recovery from loss detection

- After ½ of the *dummy* ACKs are received, the sender gains confidence; it suspects the loss was due to errors (instead of congestion)
- For the remaining ½ of the *dummy* ACKs, it increases the rate by 1/SRTT for each ACK received
- In the end, if **ALL** ACKs are received, the final rate is **equal to the rate before loss** detection

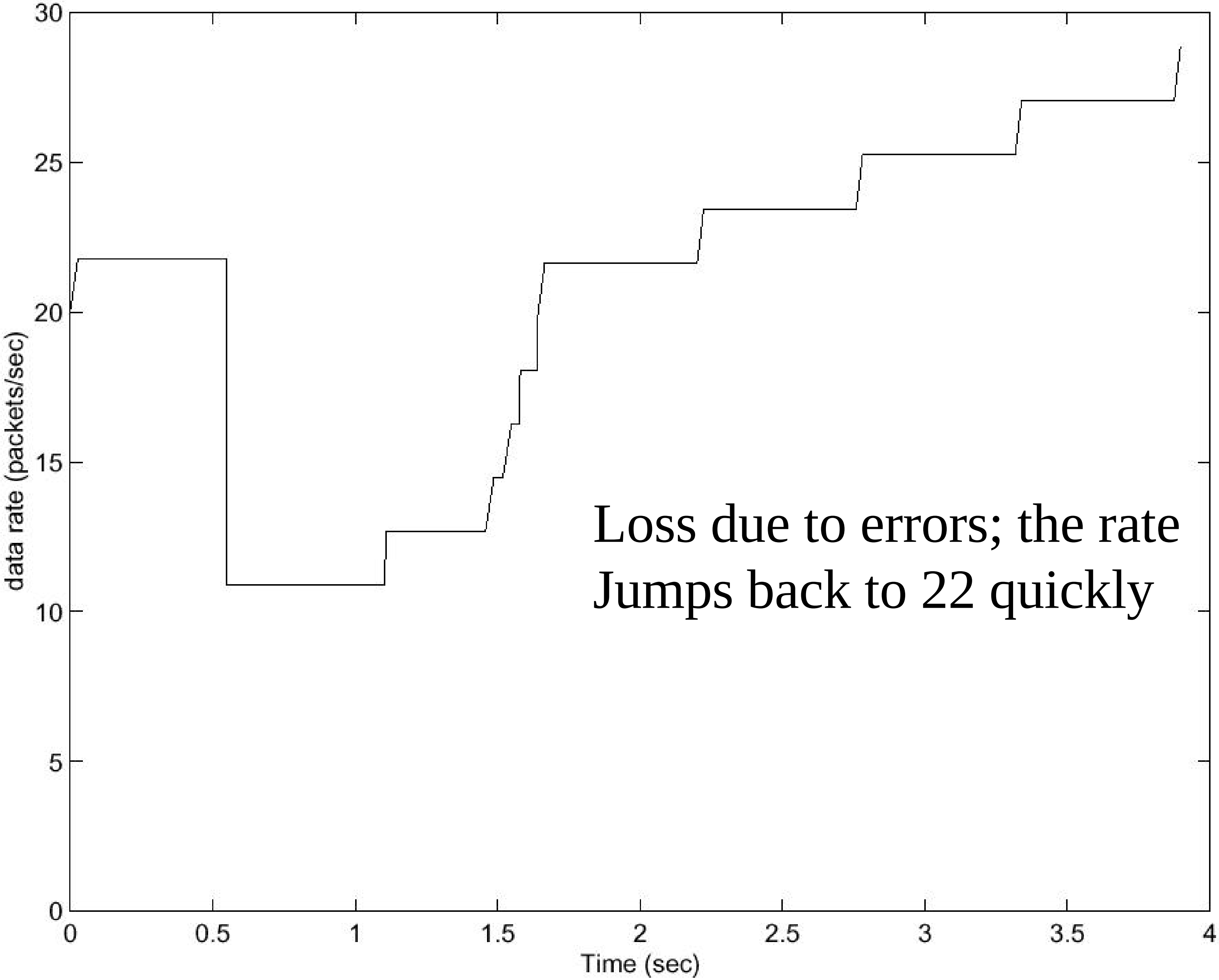
Recovery from loss detection (cont)

- If the loss is due to congestion, ½ of the dummy pkts will be dropped (the path can accept only at most a rate = S, while sender is pumping at the rate = S/2 data pkts + S dummy pkt)
- Thus, after the surviving ½ dummy ACKs have been received by the sender (best case), there are no more ACKs that allow the increase of S
- Thus, sender is stuck in the S/2 rate (as we wanted it to be, to mimic TCP in congestion loss)!

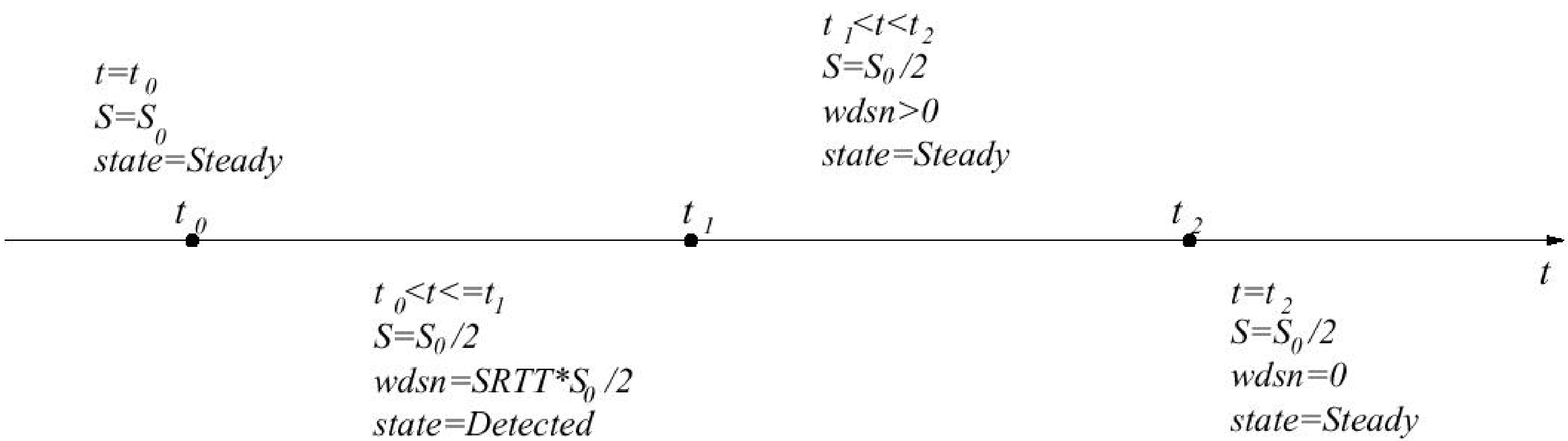
```
Detected()
    t_0 = t;
    t_{END} = t_0 + SRTT;
    S = S/2;
    IPG = 1/S;
    t_{next_data} = t_0;
    wdsn = SRTT · S;
    while (t ≤ t_{END})
        while (t ≤ t_{next_data});
        send(DATA_PACKET);
        while (t ≤ t_{next_data} + IPG/3);
        send(DUMMY_PACKET);
        while (t ≤ t_{next_data} + 2 · IPG/3);
        send(DUMMY_PACKET);
        t_{next_data} = t_{next_data} + IPG;
    end;
    state=Steady;
end.
```

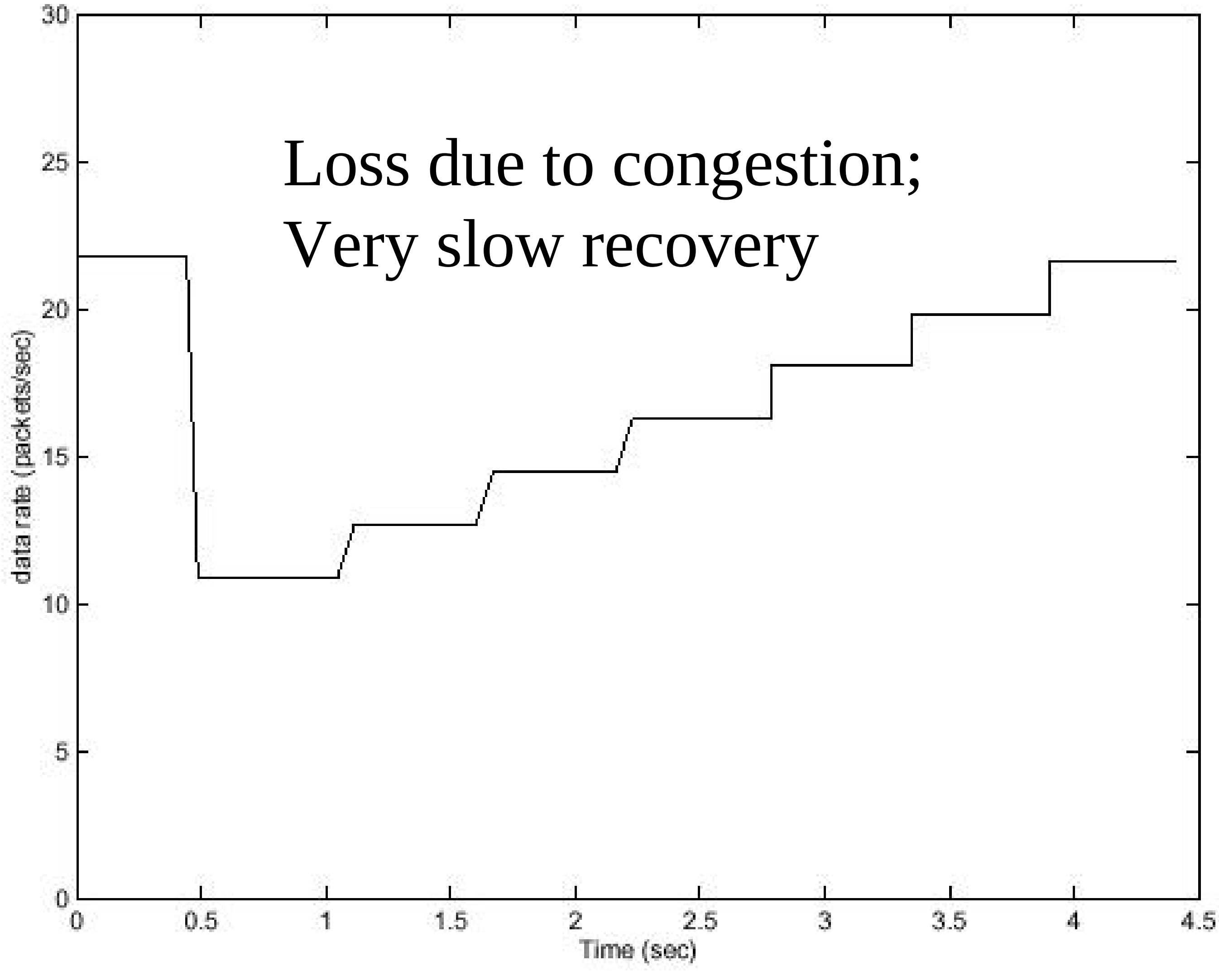


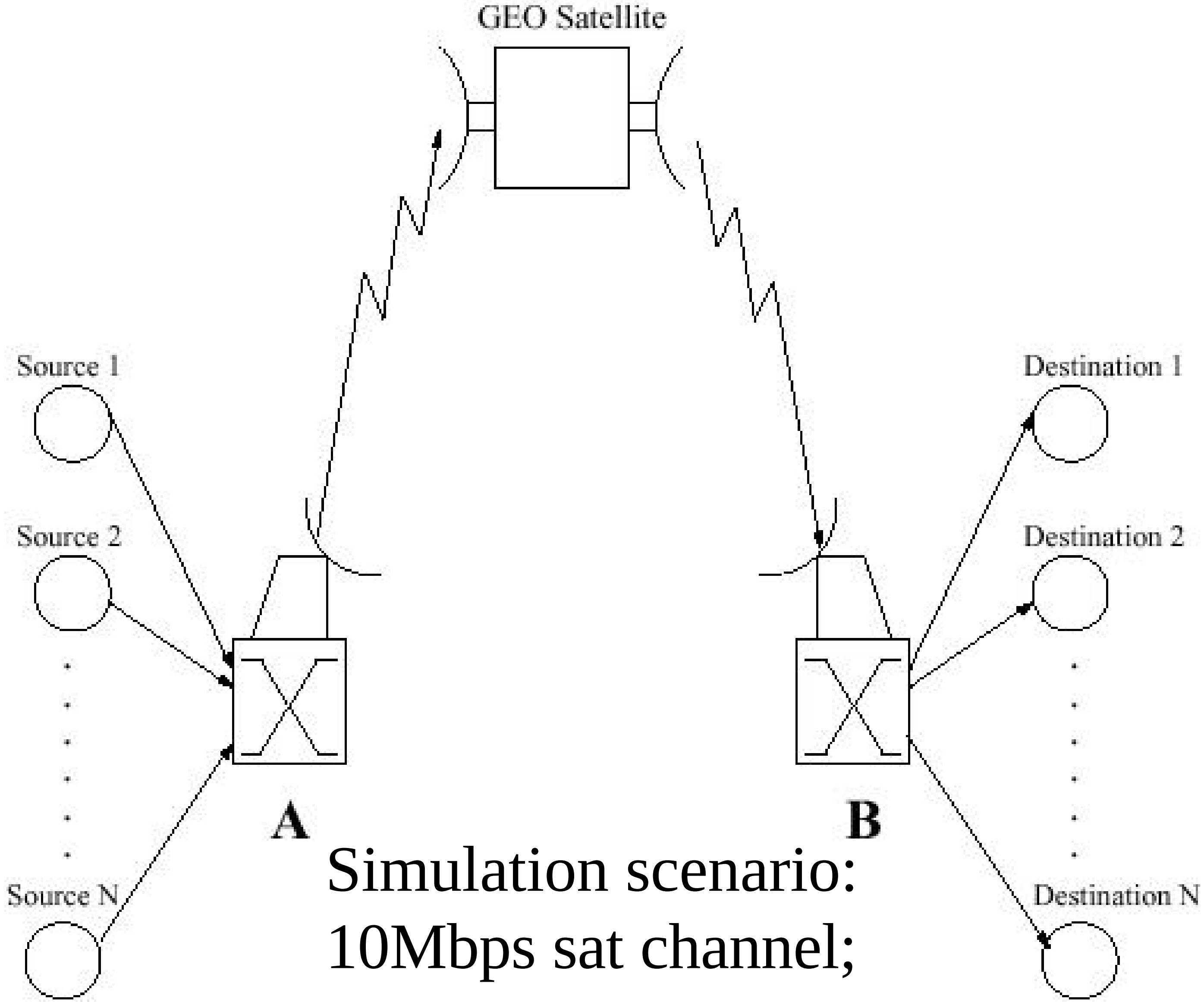
$t=t_0$
$S=S_0$
$state=Steady$
$t_0<t<=t_1$
$S=S_0/2$
$wdsn=SRTT*S_0/2$
$state=Detected$
$t_1<t<t_2$
$S=S_0/2$
$wdsn>0$
$state=Steady$
$t_2<=t<t_3$
$S_0/2<S<S_0$
$wdsn=0$
$state=Steady$
$t=t_3$
$S=S_0$
$wdsn=0$
$state=Steady$
t_0
t_1
t_2
t_3
t



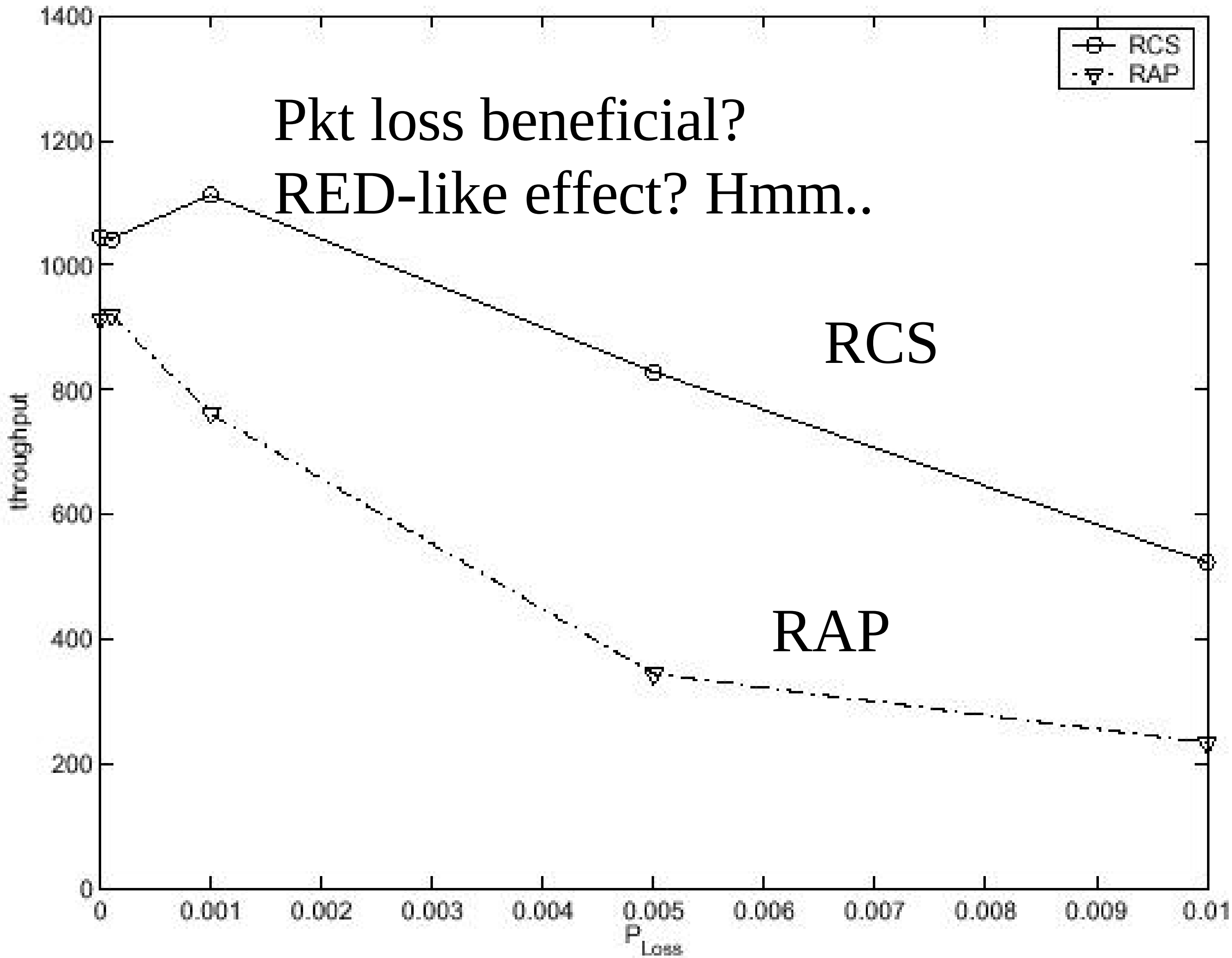
Loss due to errors; the rate
Jumps back to 22 quickly

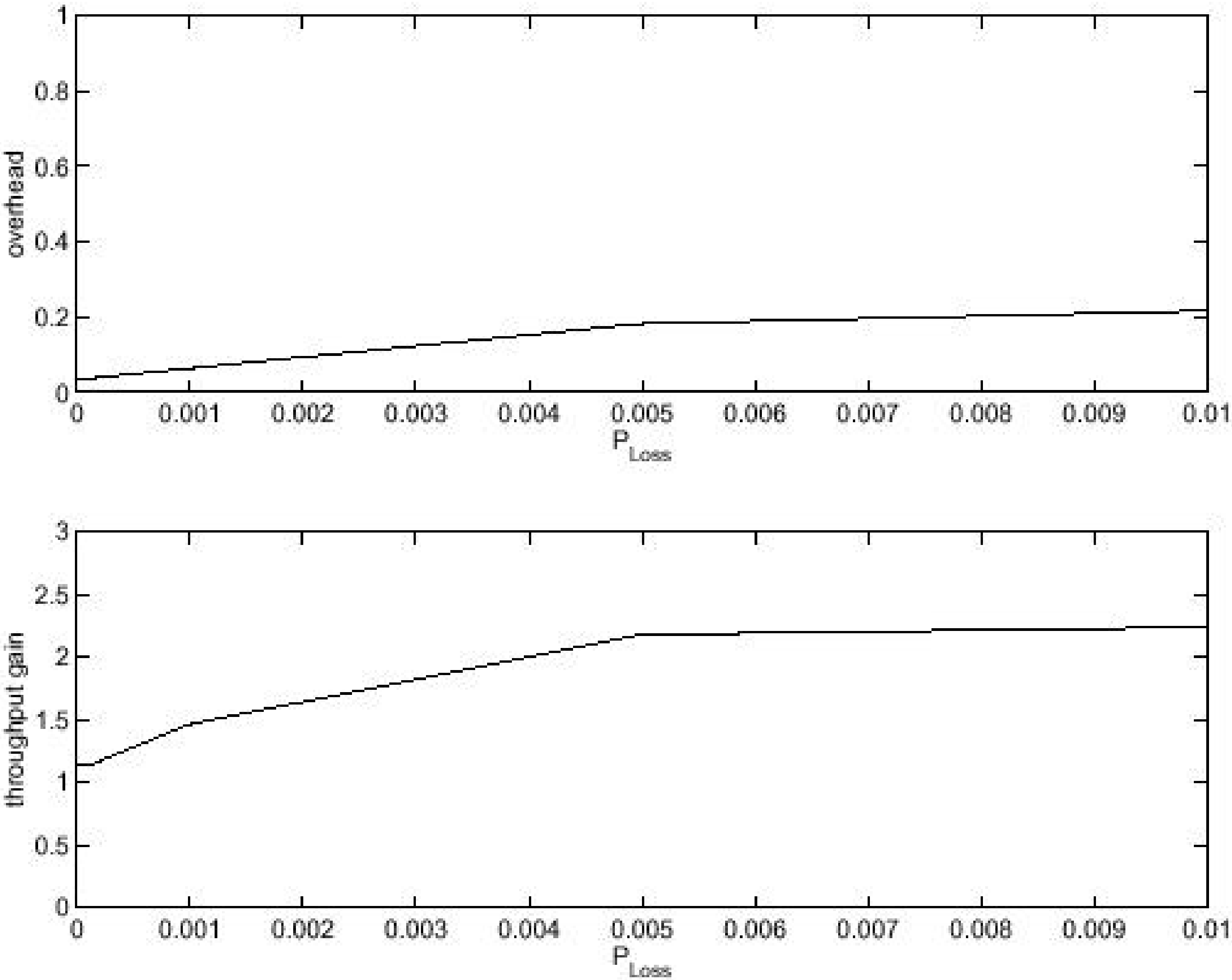

t=t 0
S=S0
state=Steady
t 1<t<t 2
S=S0/2
wdsn>0
state=Steady
t 0
t 1
t 2
t
t 0<t<=t1
S=S0/2
wdsn=SRTT*S0 /2
state=Detected
t=t 2
S=S0/2
wdsn=0
state=Steady


Loss due to congestion;
Very slow recovery
data rate (packets/sec)
Time (sec)
0
5
10
15
20
25
30
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5

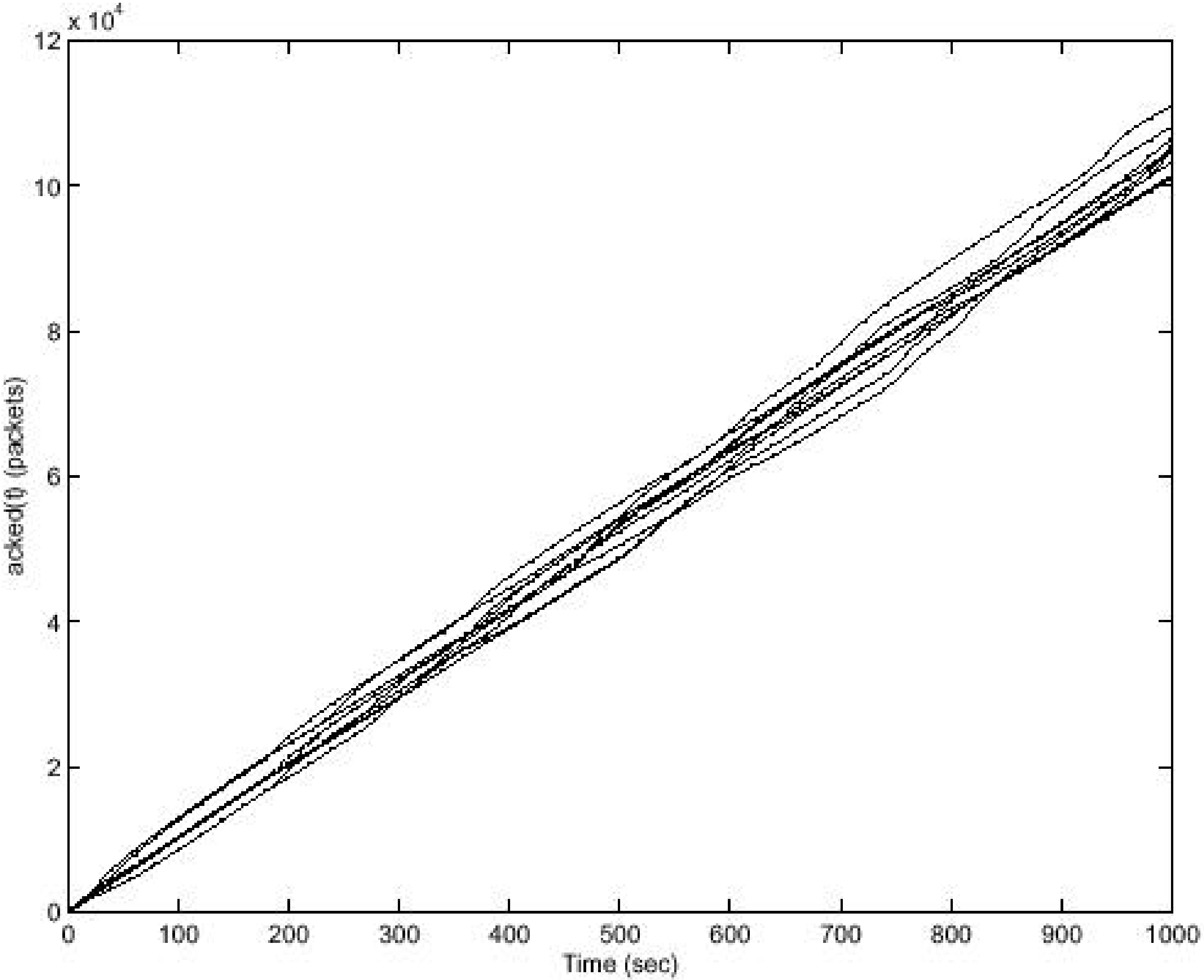


Simulation scenario:
10Mbps sat channel;
RTT = 550ms

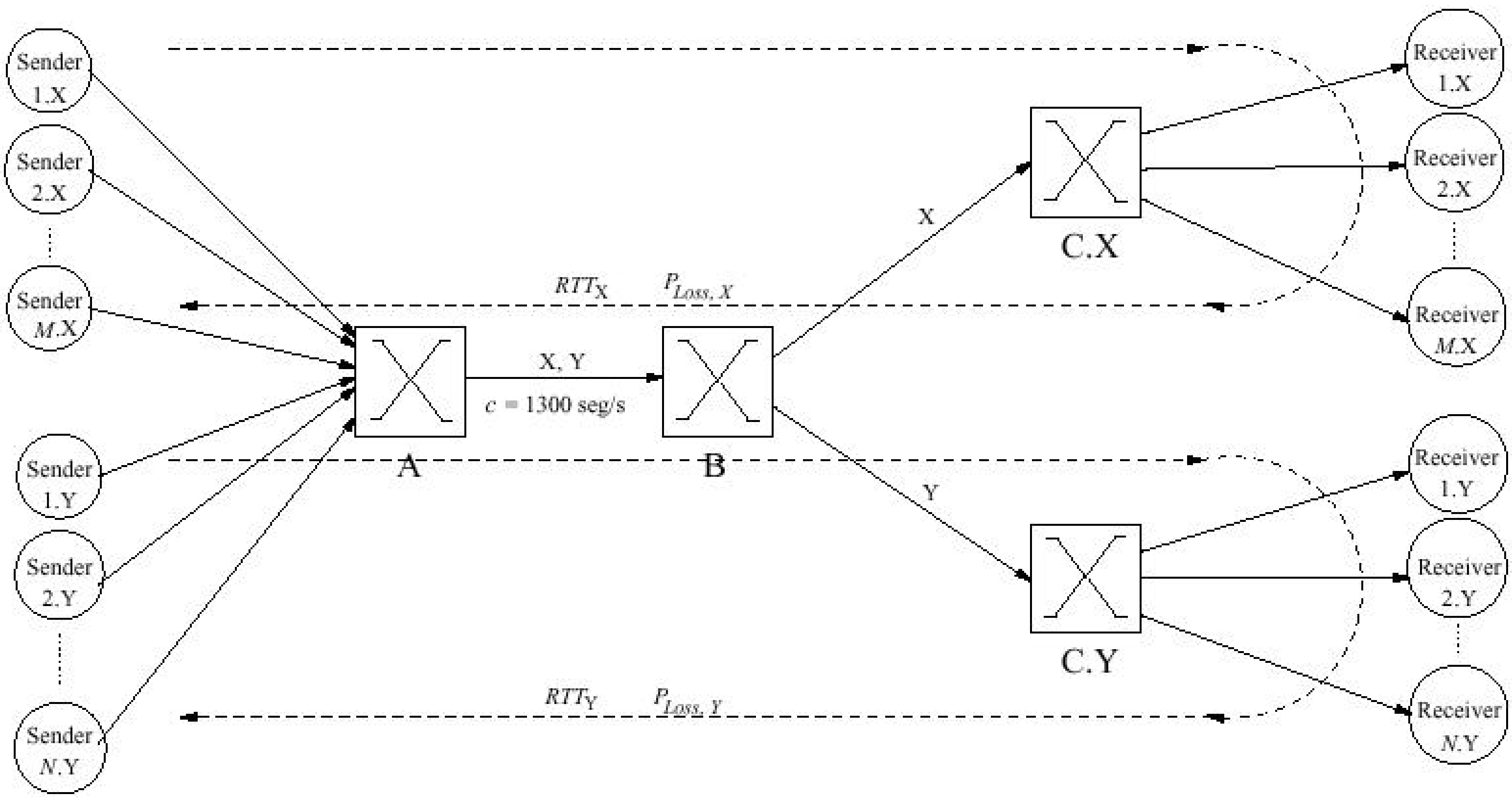

Pkt loss beneficial?
RED-like effect? Hmm..
RCS
RAP
RCS
RAP
1400
1200
1000
800
600
400
200
0
throughput
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
P Loss



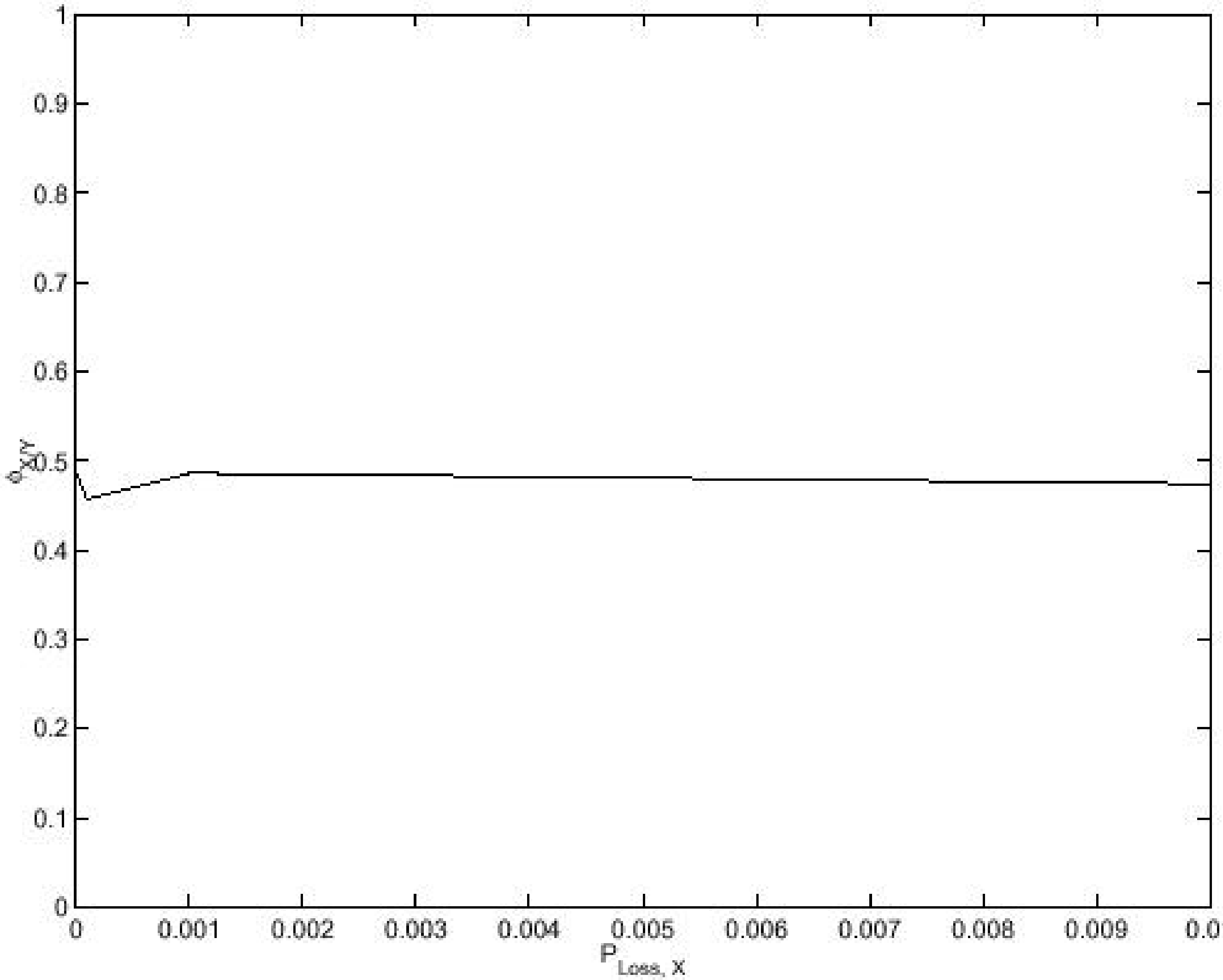
Comparison of Bdw O/H and throughput gain of RCS vs RAP



Fairness among homogeneous RCS connections

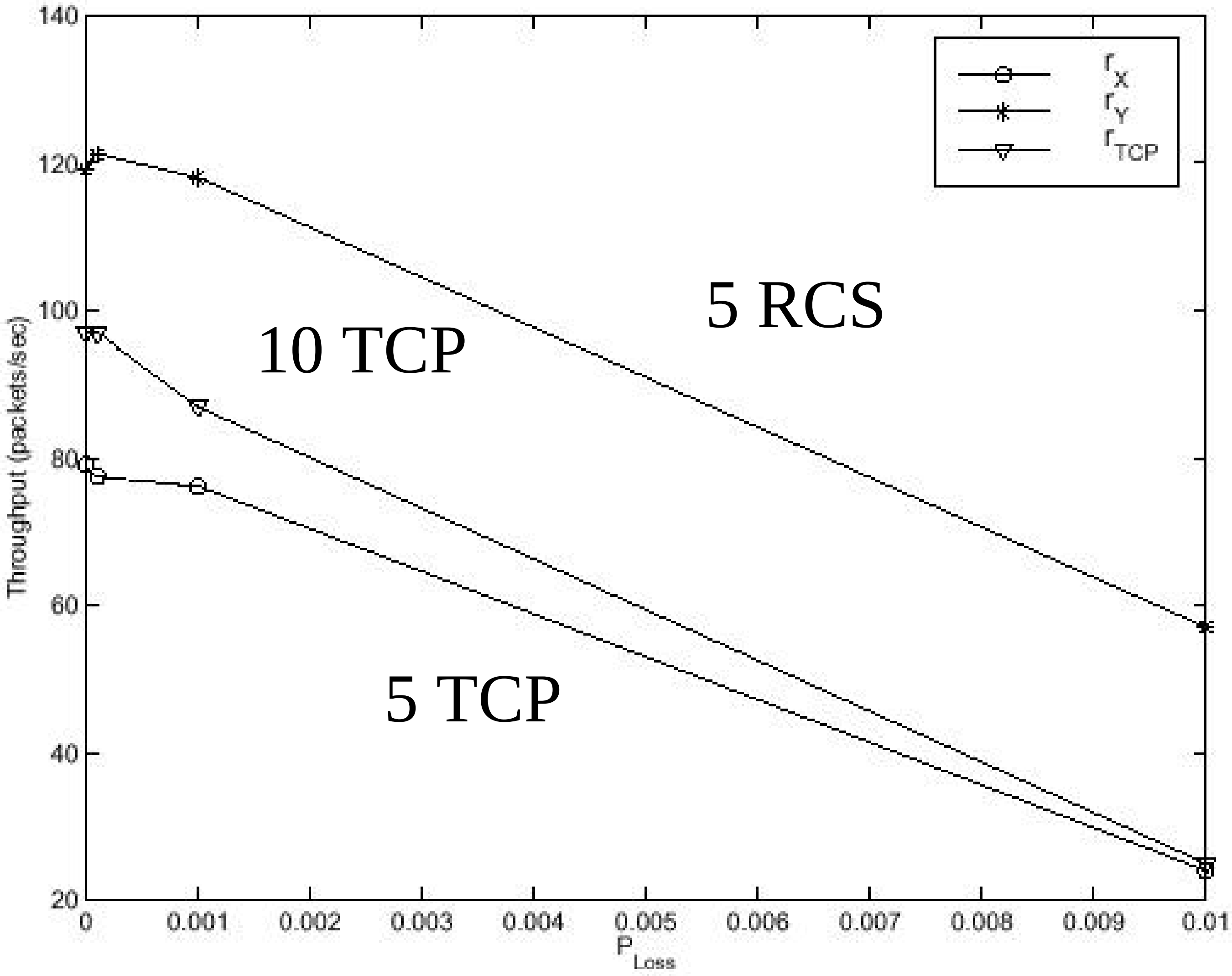


Sender 1.X
Sender 2.X
Sender M.X
Sender 1.Y
Sender 2.Y
Sender N.Y
A
X, Y
c = 1300 seg/s
B
X
Y
C.X
C.Y
RTT_X
P_Loss, X
RTT_Y
P_Loss, Y
Receiver 1.X
Receiver 2.X
Receiver M.X
Receiver 1.Y
Receiver 2.Y
Receiver N.Y



1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
$\phi_{X/Y}$
$P_{Loss, X}$

Friendliness to TCP



Compare: {5 RCS + 5 TCP} vs all 10 TCP

Conclusion

- Intriguing streaming protocol
- The probing with *dummy* pkts is clever
- Relies on existence of low priority packets (lower priority than best effort)
- Truly ETE scheme (middleware, above UDP and RTP)
- Need more Friendliness experiments to convince us of peaceful coexistence with TCP