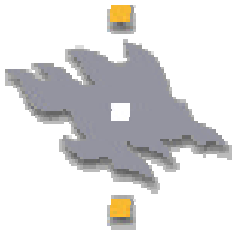


University of Helsinki - Department of Computer Science

Analysis of Internet Transport Service Performance with Active Queue Management in a QoS-enabled Network

Oriana Riva

oriana.riva@cs.Helsinki.Fl

Contents

- **TCP Congestion Control in the Internet**
- **Router queue management algorithms**
  - Tail Drop discipline
  - Active Queue Management (AQM)
    - Goals
    - Random Early Detection (RED)
- **Test Arrangements**
  - Test Environment
  - Test Cases
- **Test Results and Analysis**
- **Conclusions**

TCP Congestion Control in the Internet

- Internet has grown and TCP congestion control mechanisms are not sufficient to provide good service in all circumstances
  - Wired vs. Wireless
  - The Internet is evolving to support QoS
- Problem: There is a limit to how much control can be accomplished from the edges of the network
- A possible solution is an advanced form of router queue management such as Active Queue Management (RFC 2309)
- Current router algorithm employed in the Internet: Tail Drop

Tail Drop buffer

- **Principle:**
  - fix the maximum length for each queue, accept packets for the queue until the maximum length is reached, then drop subsequent incoming packets
- **Advantages**
  - easy to implement
- **Disadvantages**
  - Full Queues
    - Discriminate against bursty traffic and cause multiple packets to be dropped
    - Synchronization of sources
  - Lock-Out: a single connection or few flows manage to monopolize the available queue resources

Active Queue Management (AQM)

- **Based on a proactive approach:**
  - drop packets before buffer becomes full
  - use the average queue length as congestion indicator
- **Can provide the following advantages for *responsive* flows**
  - Control the average queuing delay and reduce end-to-end latency
  - Guarantee more equity in resources utilization (fairness)
  - Reduce the number of packets dropped in routers
  - Provide greater capacity to absorb bursts without dropping packets
  - Avoid global synchronization of sources
  - Avoid Lock-Out behaviour
- **The most famous example of AQM is the Random Early Detection (RED) algorithm**

RED - Algorithm

- Calculate the average queue size (*avg*) by a low pass filtering with weight w_q, $0<w_q<1$

$$avg_{n+1} = (1 - w_q)avg_n + w_q q_{ist}$$

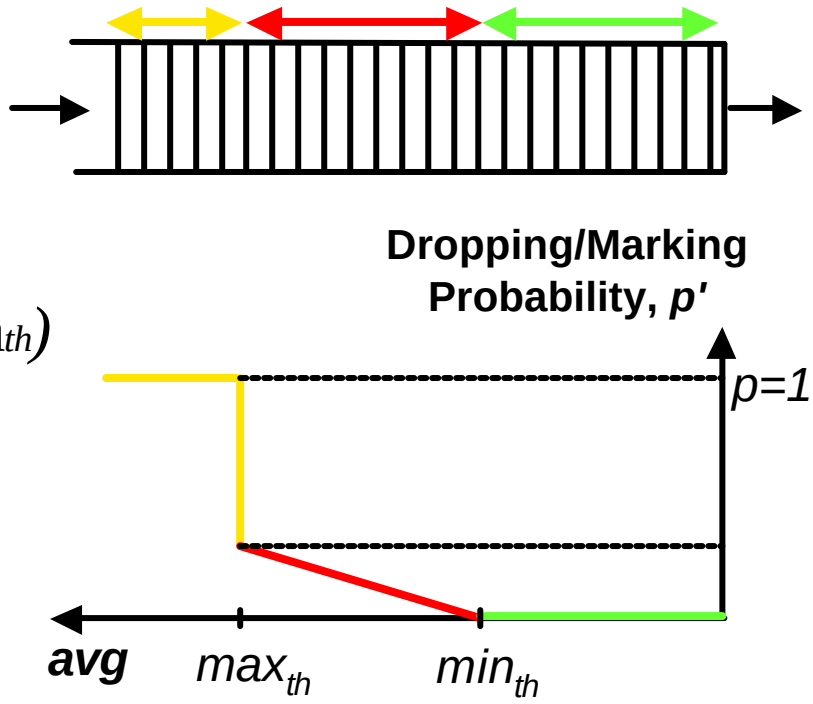
q_{ist}=*instantaneous queue length*

- If $avg<min_{th}$ do nothing
- If $avg>max_{th}$ drop the packet
- Else drop (or mark) packet with probability:

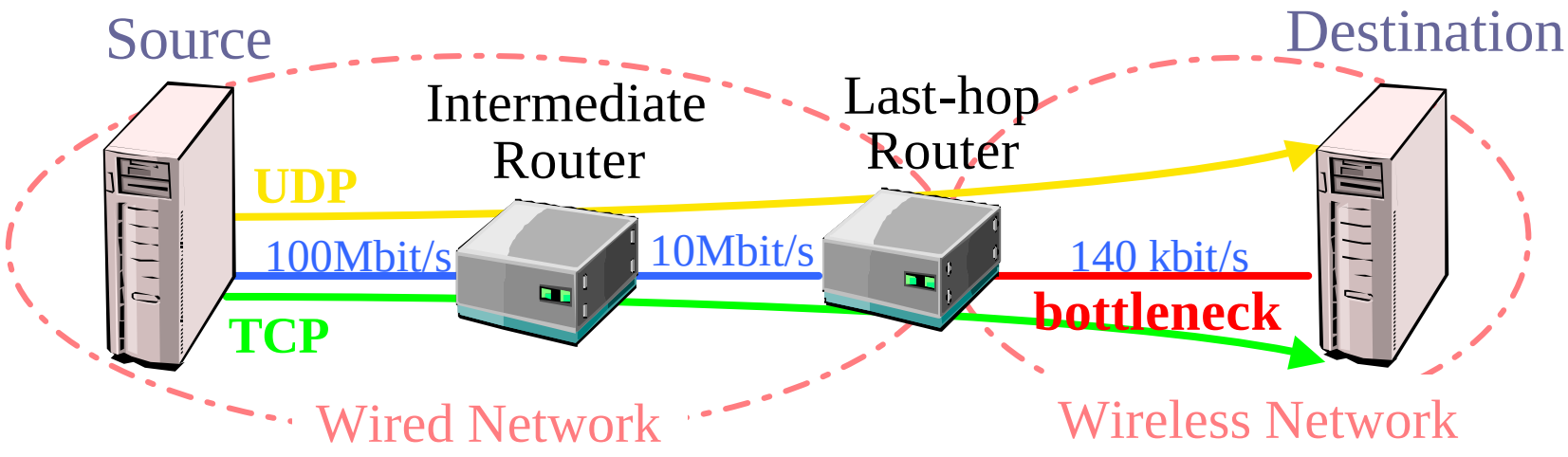
$$p' = \max_p (avg - \min_{th}) / (\max_{th} - \min_{th})$$

$$p = p'/(1 - count \cdot p')$$

count =number of unmarked packets since the last time a packet was dropped or since avg exceeded min_{th}



Test Arrangements

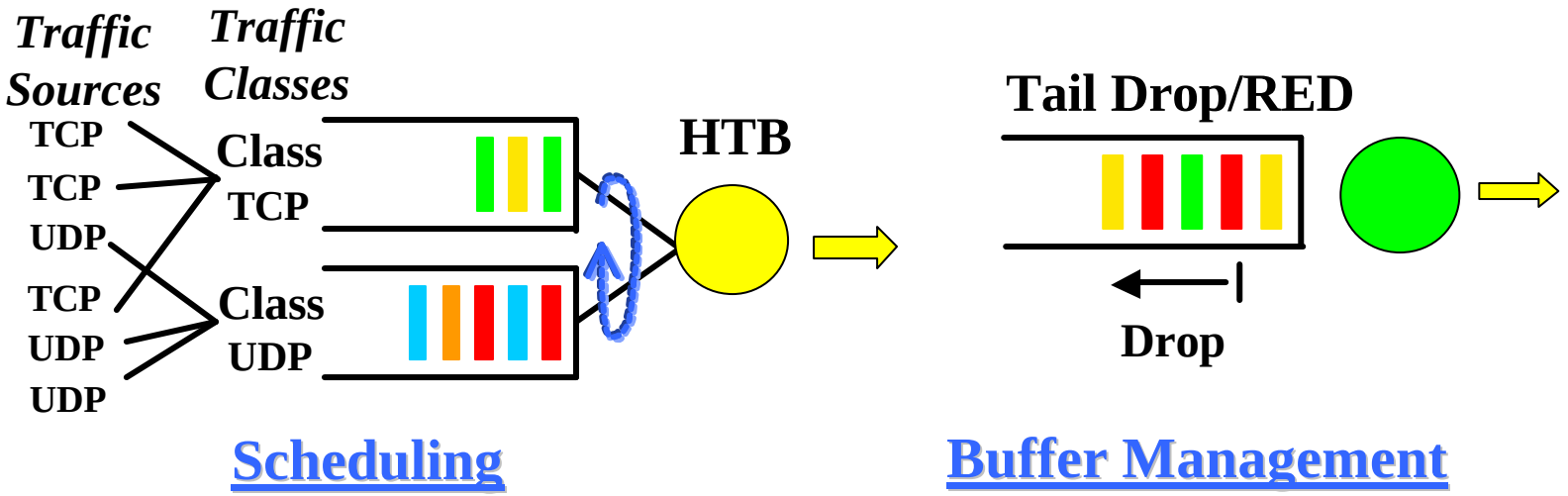


Simple scenario (Each scenario is tested with 20 replications)

- PC with Linux Operating System (kernel version 2.4.17)
- Standard Linux TCP Implementation with SACK option (Timestamps: OFF
- The Intermediate Router employs Tail Drop with a huge buffer size
- The Last-hop Router employs Tail Drop or RED without ECN
- RED Parameters: buffer size=20kbytes, min_{th}=5kbytes, max_{th}=18kbytes, max_p=0.1
- Metrics: Elapsed Time, Num Retx Packets, Fairness, etc..

Test Arrangements - Test Environment

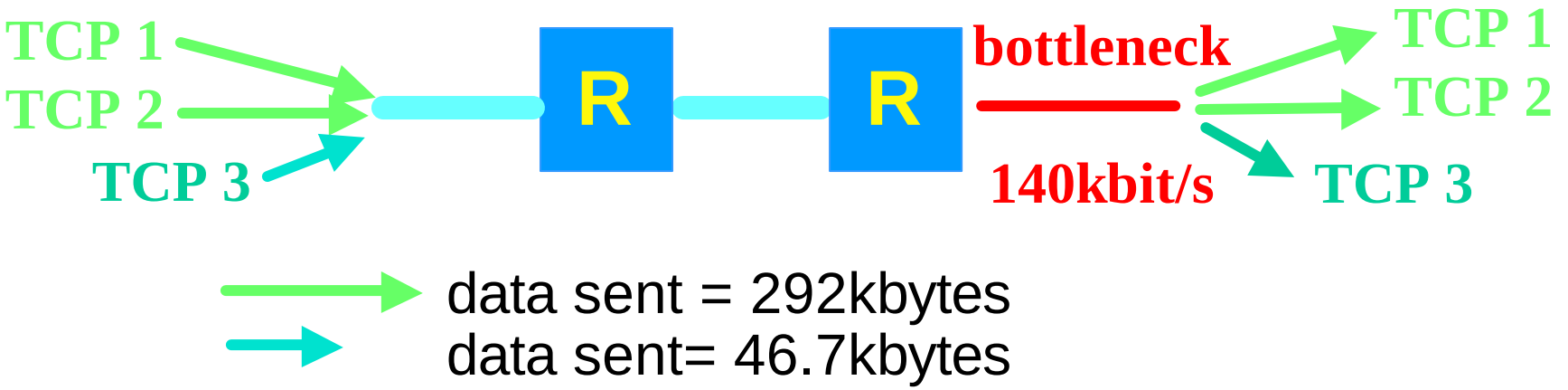
- The last-hop router implements a *DiffServ* (DS) PHB using HTB *(Hierarchical Token Bucket)* packet scheduler with two service classes
- Each service class employs Tail Drop or RED
- Queuing allocates bandwidth and buffer space:
  - Bandwidth: which packet to serve next (scheduling)
  - Buffer Space: which packet to drop next (buffer management)



Test Arrangements - Test Case 1

Workload: 3 competing TCP flows

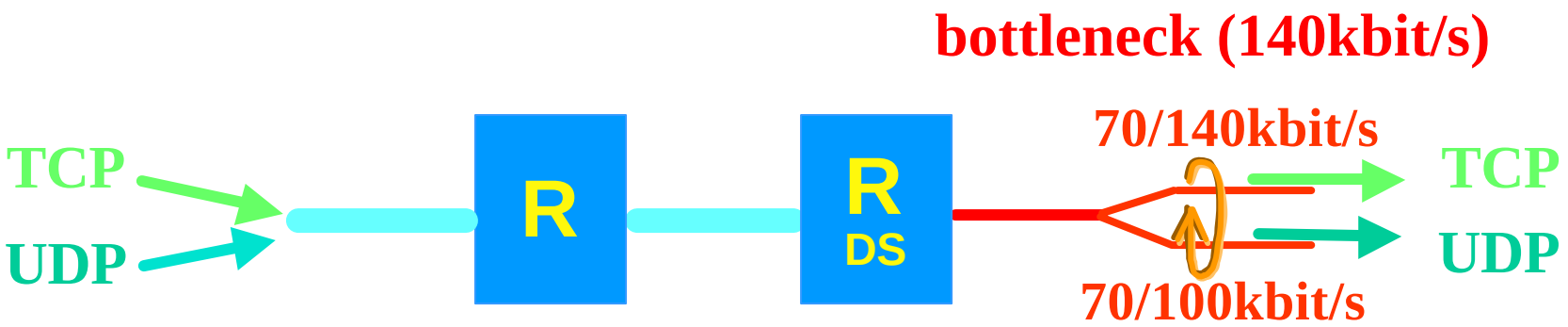
- 1 service class – bottleneck bandwidth =140kbit/s
- 2 TCP connections start simultaneously (data traffic sent=292kbytes)
- The third connection starts with a delay varying from 0 up to 7 seconds (data traffic sent= 46.7kbytes)



Test Arrangements - Test Case 2

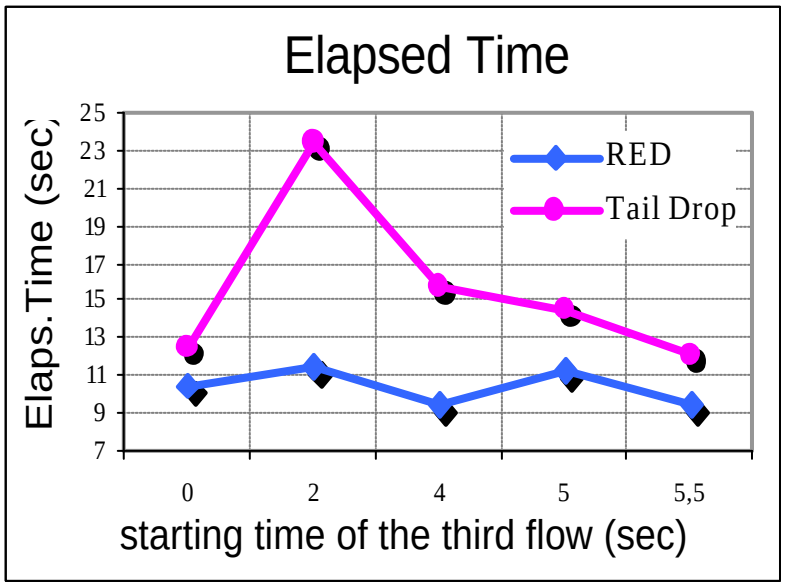
Workload: TCP and UDP traffic with DiffServ

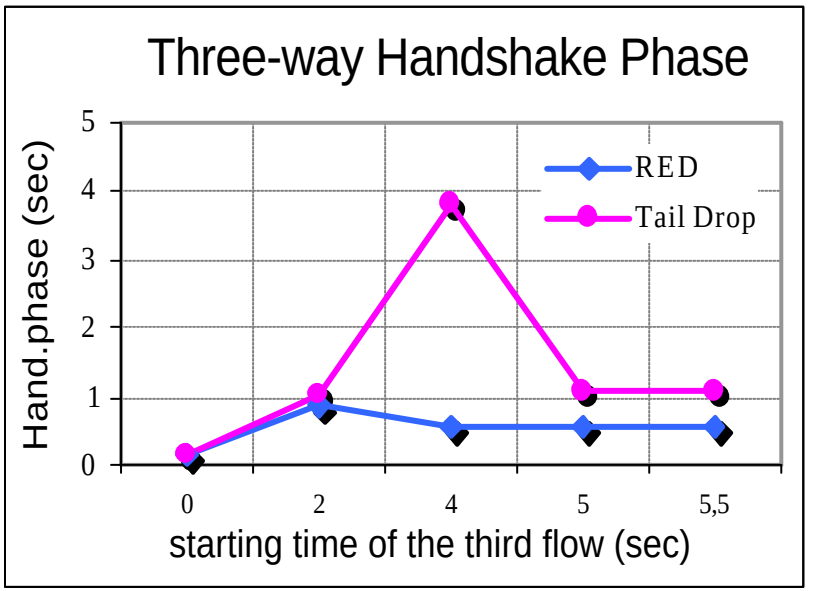
- The UDP traffic starts later and has different duration
  1. The UDP traffic lasts for the entire duration of the TCP transfe
  2. The UDP traffic ends before the TCP connection ends
- Effects derived from the introduction of service differentiation:
  1. Unique service class for TCP and UDP traffic (available bandwidth=140kbit/s)
  2. Separate service classes for TCP traffic and higher-priority UDP traffic (class bandwidth=70kbit/s)



Results – Test Case 1

- 3 competing TCP connections
- Results of the third connection for different starting points



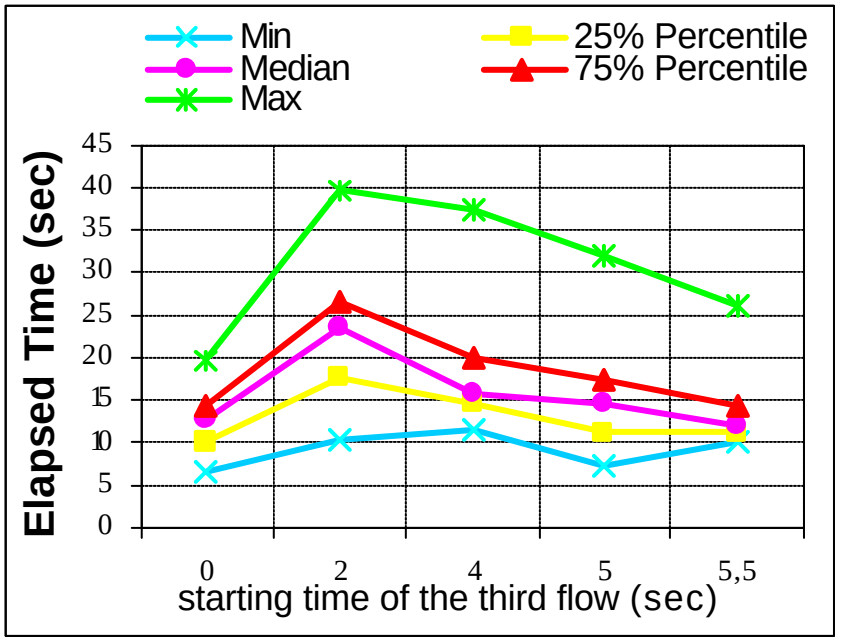


- Stable results and improved performance when RED is employed
- Tail Drop
  - Scenario *after 2 seconds*: the elapsed time doubles the RED's one!

Results – Test Case 1

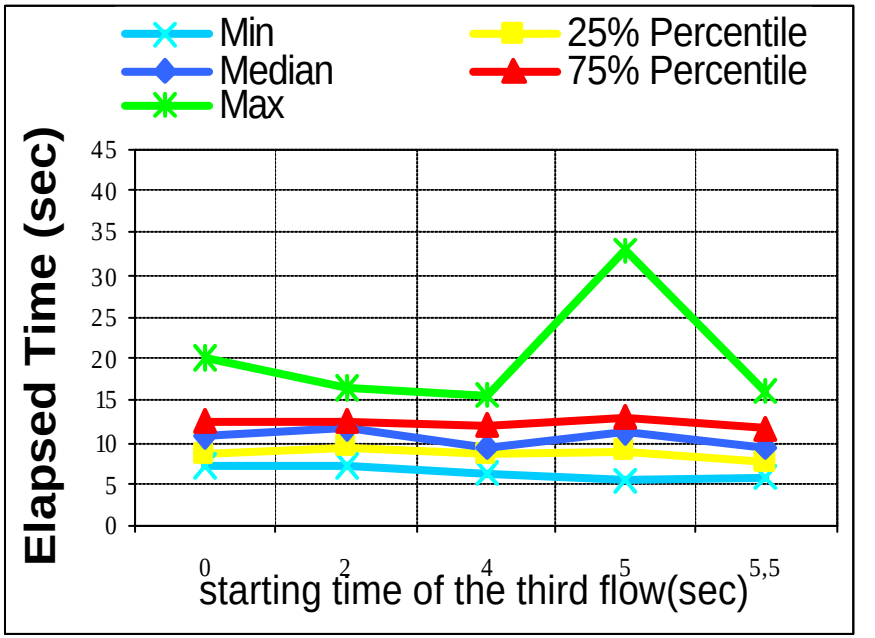
3 competing TCP connections – Elapsed Time of the 3rd flow

Tail Drop



- **Wide variation range**

RED

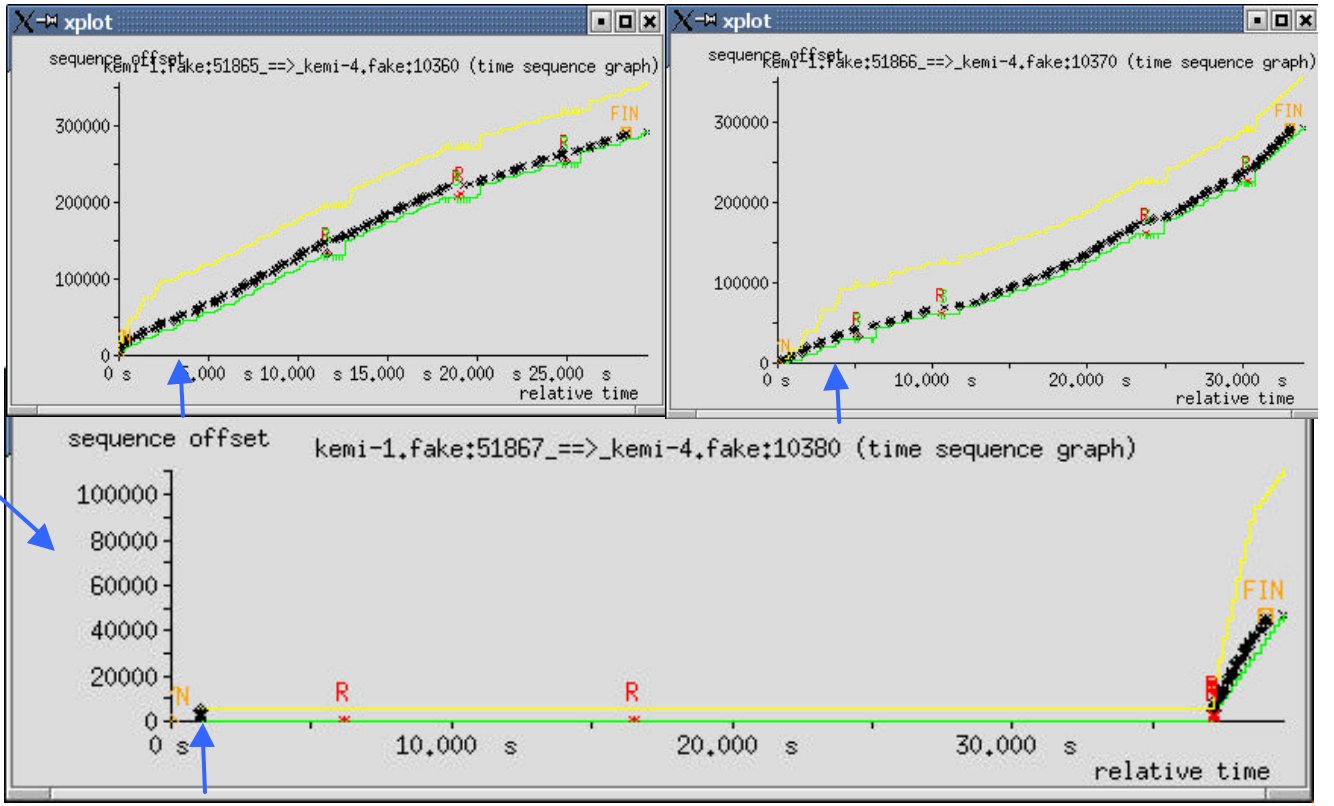


- **Limited variation range**

Test Case 1 - Lock-Out of Tail Drop

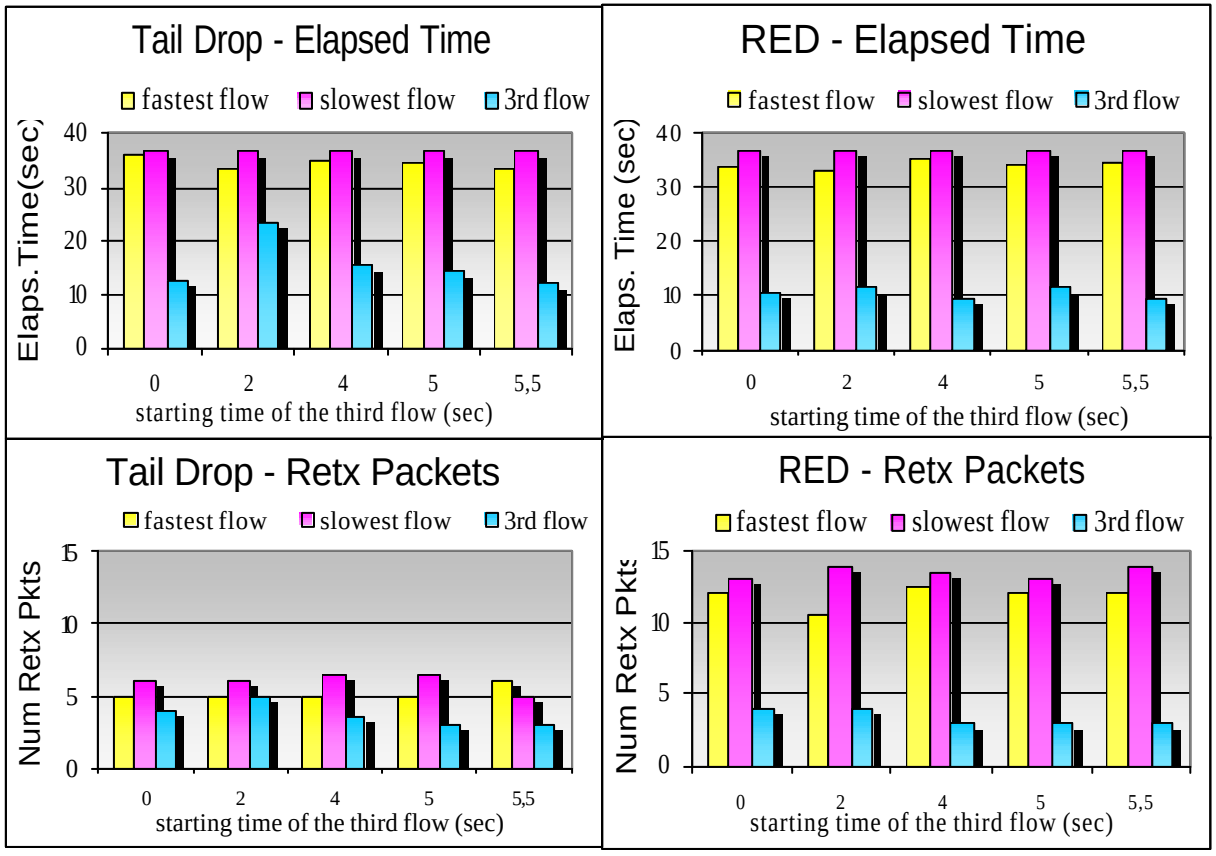
- Tail Drop: example from the scenario *after 2 seconds*
- The initial flows monopolize most of the bandwidth (phenomenon of Lock-Out)

The exponential retransmit back-off rule doubles the retransmit timer when a retransmitted packet is lost



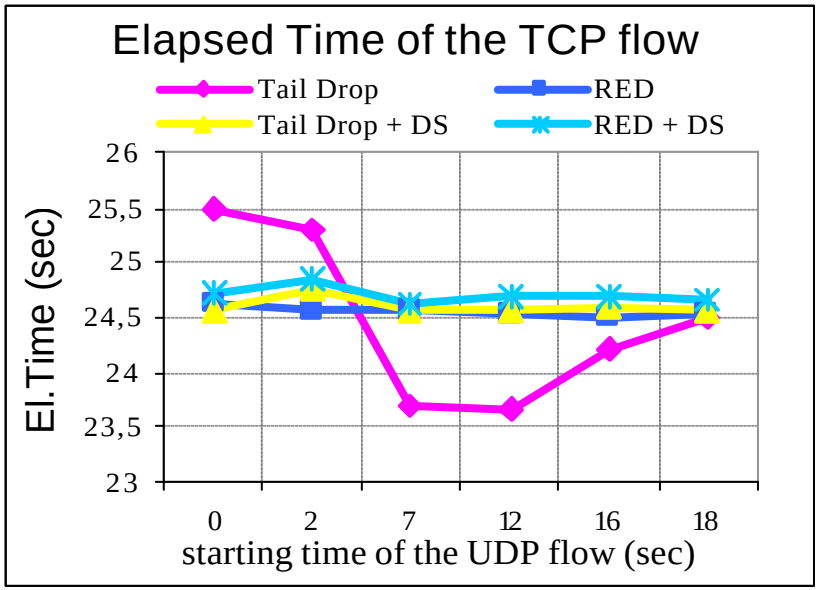
Results - Test Case 1

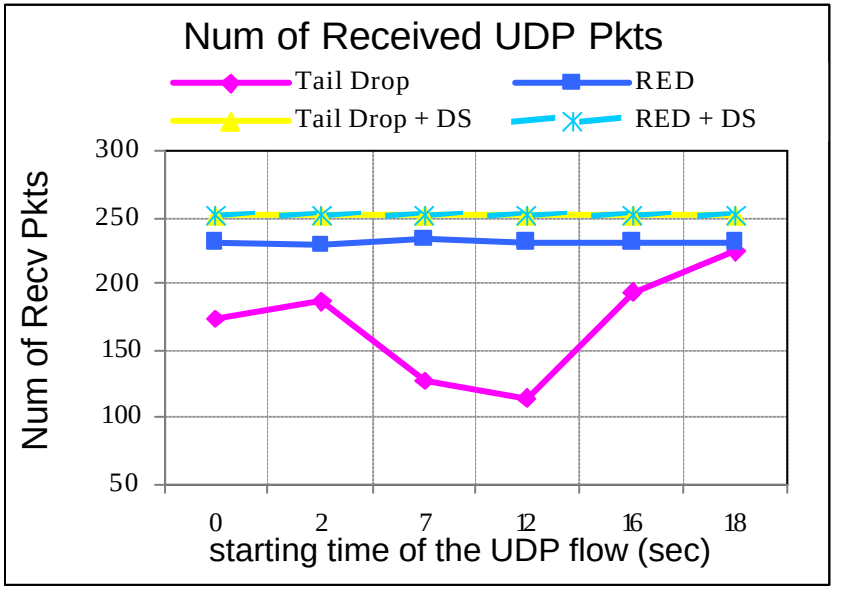
- 3 competing TCP flows
- Third flow's data load is 4/25 of the initial flows' data load
- RED guarantees to the third flow improved and stable performance
- RED drops packets in proportion to the amount of bandwidth the flow uses on the output link



Results - Test Case 2

- Single TCP flow competing with UDP traffic lasting 5sec

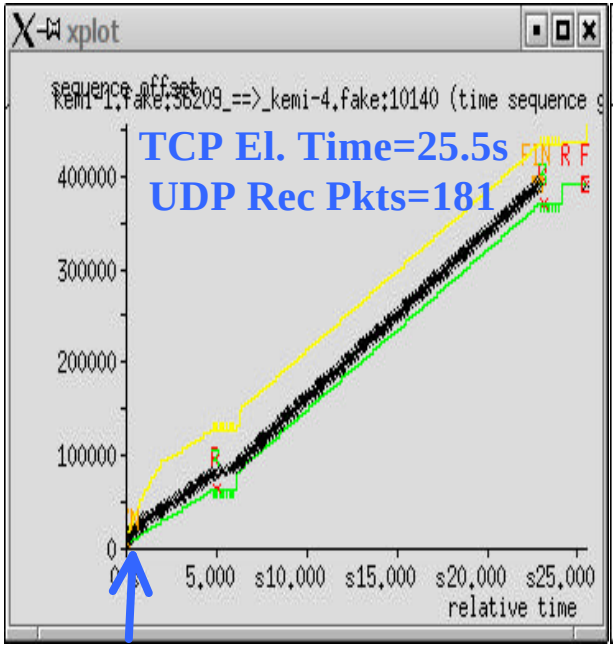


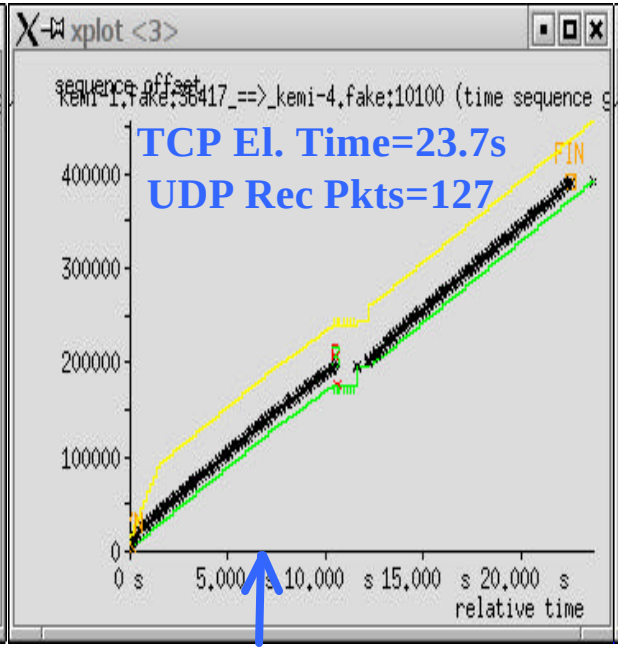


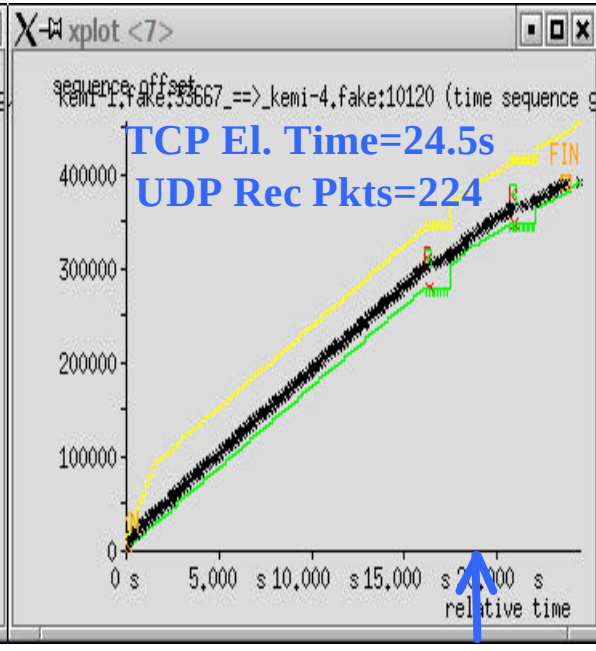
- 1 service class :
  - Tail Drop: unstable results for both TCP and UDP traffic
  - RED: performance close to the results with 2 service classes
- 2 service classes (Higher priority UDP traffic)
  - Tail Drop and RED achieve the same performance

Test Case 2 - Unfairness of Tail Drop

- TCP traffic competing with UDP traffic of equal priority
- The performance changes by varying the starting point of the UDP traffic



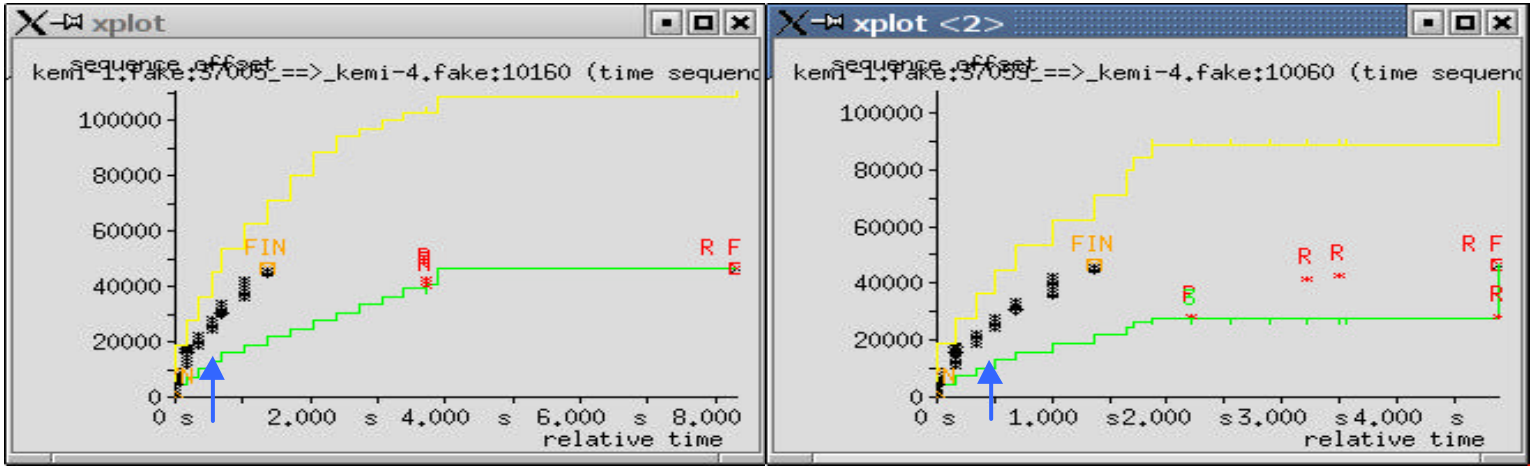




UDP flow starts

Results – Test Case 2

- Time Sequence Graph of Traffic TCP competing with higher-priority UDP traffic starting after 0.5s



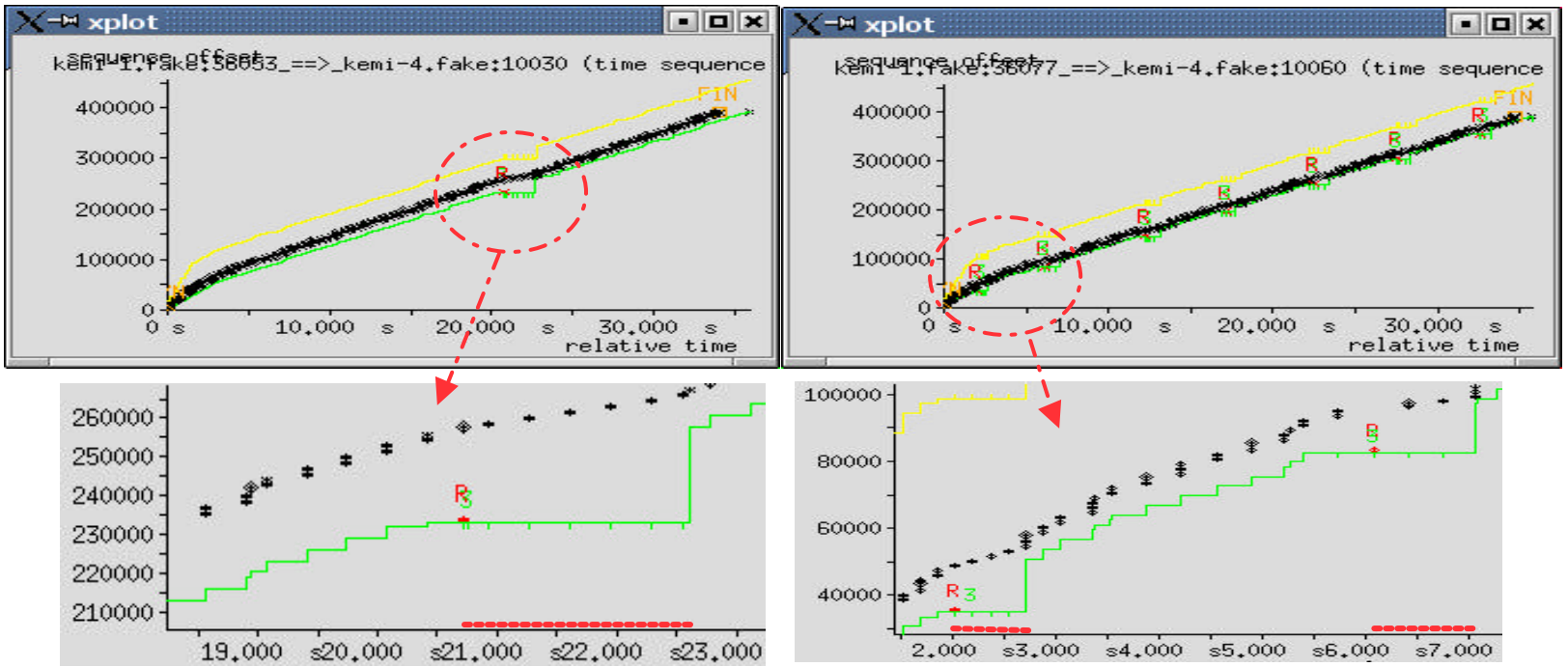
- Tail Drop: el. time=8.3s
- Timeout expiration
- Start to drop packets late
- Discard consecutive packets

- RED: el. time=4.9s
- 3 DUPACKs
- Start to drop packets early
- Discard randomly + SACK option

Results – Test Case 2

- 1 TCP flow + UDP traffic (starting after 3 seconds) in presence of service differentiation

Tail Drop: el.time=35.9s; rtx pkts=1 **RED**: el.time=35,8s; rtx pkts=7

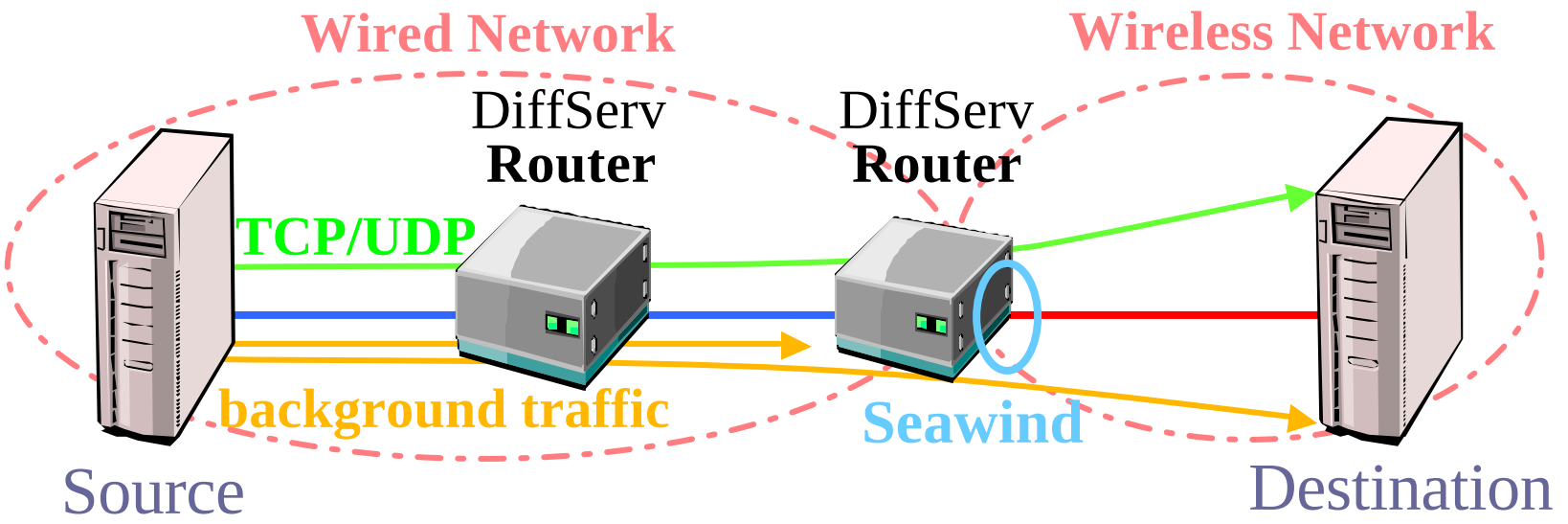


- The dropped packets are evenly distributed over a wide range
- The Recovery phase with Tail Drop is roughly the double of the RED's one

Conclusions

- **RED fairness**
  - Fair sharing of resources
  - Avoid Lock-Out
  - Stable results
- **RED drops more packets than Tail Drop, BUT**
  - By starting to drop packets early it manages to recover quickly from congestion and provides improved performance
  - It drops packets from each flow in proportion to the amount of bandwidth the flow uses on the output link

Future Work



- Emulate various wireless link characteristics on the last-hop link
- Add background traffic
- Advanced use of DiffServ with different traffic mixes
- Use of TCP variants
- Use of ECN (*Explicit Congestion Notification*) in combination with RED
- Tuning of RED parameters