RED+ GATEWAYS FOR IDENTIFICATION AND DISCRIMINATION OF UNFRIENDLY BEST-EFFORT FLOWS IN THE INTERNET

Thomas Ziegler[†‡], Serge Fdida[†], Ulrich Hofmann[‡] [1]

[†] *Université Pierre et Marie Curie, Laboratoire Paris 6, Paris, France*

[‡] *Polytechnic University Salzburg, School for Telecommunications, Salzburg, Austria*

{Thomas.Ziegler, Serge.Fdida}@lip6.fr, {Thomas.Ziegler, Ulrich.Hofmann}@fh-sbg.ac.at

Abstract -- This paper proposes an add-on to the well known RED (Random Early Detection) algorithm called RED+. RED+ adds the functionality of identifying and discriminating high-bandwidth, unfriendly best-effort flows to RED gateways. It is based on the observation that unfriendly flows have higher arrival rates in times of congestion than friendly flows. Hence unfriendly flows can be discriminated if packets arriving at a router output-port are dropped as a function of RED's average queue size and the arrival-rate of the packet's flow. RED+ is scalable regarding the amount per flow information stored in routers as it only allocates per-flow state for the n highest bandwidth flows, where n is a configurable parameter. As shown by simulation, RED+ is able to identify and discriminate unresponsive flows avoiding problems of unfairness and congestion collapse.

Keywords: Congestion Control, unfriendly Flows, RED Gateways, max-min Fairness

1 INTRODUCTION

So called "unfriendly flows" reduce their sending-rate into the net less conservatively in response to congestion-indications (packet-loss) than friendly flows[2]. Consequently, unfriendly flows tend to grab an unfairly high portion of the bottleneck-bandwidth. "Unresponsive flows" exhibit an extreme kind of unfriendly behavior as they do not back-off at all in response to congestion indications. Simulations in [7] show that for a friendly and an unresponsive flow sharing a link, throughput of the friendly flow and arrival-rate of the unresponsive flow are inversely proportional. In other words, throughput of the friendly flow converges to zero if a non responsive flow's arrival-rate approaches the link-bandwidth. This behavior strongly violates the goal of fair distribution of bandwidth among best-effort flows. Additionally, unresponsive flows can cause congestion collapse due congested links transmitting

---

[1] This work is partly sponsored by Telecom Austria and FFF Austria.

[2] A flow is defined by IP address pair and port-numbers, respectively flow-ids.

packets that are only dropped later in the network [7].

For the Internet, TCP flows can be considered friendly as TCP congestion control in its different derivates exhibits roughly homogeneous behavior. Due to the dominance of TCP in today's Internet it is reasonable to define "friendly" as "TCP-friendly" [7]. However, this definition of friendliness may not hold for the future in case of widespread deployment of alternative congestion-control mechanisms.

The objective of this paper is to propose a queue-management mechanism called RED+ adding the functionality of identifying unfriendly flows to RED gateways [8]. Using RED as a basis, RED+ inherits desirable properties of RED like controllable queueing-delay, avoidance of global synchronization and avoidance of a bias against traffic-bursts. Note, however, that this add-on for identification of unfriendly flows is rather orthogonal from a specific queue-management algorithm as it solely requires a packet-drop function which is monotonically increasing with the queue-size and a facility for preferential packet-dropping. Hence the mechanism proposed in this paper could be used in combination with other queue-management algorithms than RED.

Due to the current lack of mechanisms like RED+ users have an incentive to be misbehaving and to generate unfriendly flows obtaining a higher share of the bottleneck bandwidth. Contrary, the deployment of mechanisms identifying and discriminating unfriendly flows would encourage users to utilize conforming end-to-end congestion control.

2 RELATED RESEARCH

The Random-Early-Detection (RED) algorithm [8] employs the parameter-set {*minth, maxth, maxp*} in order to probabilistically drop packets arriving at a router output-port. If the average queue-size (*avg*) is smaller than *minth* no packet is dropped. If $minth < avg < maxth$, RED's packet-drop-probability varies between zero and *maxp*. If $avg > maxth$, each arriving packet is dropped. In order to take into account flows with different packet sizes, RED can be operated in "byte-mode" weighting the drop-probability by the incoming packet's size. WRED [4] and RIO [5], both enhancements of RED intended for service-differentiation in the Internet [2], relate arriving packets to the parameter-set $\{minth_{in}, maxth_{in}, maxp_{in}\}$, respectively $\{minth_{out}, maxth_{out}, maxp_{out}\}$ if the packet has been marked as *in*, respectively *out* according to its flow's service-profile at a network boundary. Assuming $minth_{in} \geq maxth_{out}$, in-profile packets are accommodated while out-of-profile packets have a drop-probability of one if $avg > maxth_{out}$. Hence out-of-profile packets are discriminated against in-profile packets. As opposed to WRED, which uses one average queue size for all packets in the queue, RIO computes an extra average queue-size only for in-profile packets.

In [7] routers execute a low-priority background task in periodic time-intervals. Unfriendly flows are identified as "non TCP friendly", "unresponsive" or "high bandwidth in times of congestion". [22] shows that the TCP-friendly test, as proposed in [7], is inaccurate as routers generally do not have knowledge of the end-to-end RTT. Hence unfriendly flows are unlikely to be detected by this test. Flows identified as unfriendly are discriminated by reclassification into a lower priority queue. Note that reclassifying flows from one queue into another implies the caveat of packet misordering possibly causing TCP fast retransmits and reduction of the congestion window in case more than three packets arrive out-of-order [14] [15].

The FRED algorithm [16] uses per-active-flow accounting and preferentially drops packets of flows having either more packets than a fair-share of the buffer-size stored or an outstanding-high number of packet drops. It has been shown in [22] that FRED is not able to restrict unresponsive flows to the fair-share in rather general scenarios. Additionally, per-active flow accounting means using very small time-scales enabling a bias against bursty traffic.

[24] proposes an architecture in the context of the diff-serv [2] for identification and discrimination of non-TCP-friendly flows. The principle is to detect non-TCP-friendly flows at the ingress-router by comparing arrival rates to equivalent TCP-friendly rates. If a flow is identified as non-TCP-friendly, its packets are marked as “unfriendly”. Core routers discriminate packets marked as unfriendly with RED-based drop-preference mechanisms. The RTT is measured by means of a protocol between ingress- and egress routers hence the TCP-friendly test is significantly more accurate than in [7]. Additionally, [24] only requires storage of per-flow state and a flow-lookup in ingress- and egress routers and not in core-routers.

Another idea to identify unfriendly flows has been proposed in [19]. A alternative approach to identification of unfriendly flows is to allocate a fair-share to each flow (see [1][12][13][18][22] and others). However, merely restricting unfriendly flows to their fair share does not necessarily create an incentive for users to implement end-to-end congestion control. Additionally, many unresponsive best-effort flows restricted to their fair share may still cause congestion collapse as shown in [7].

3 RED+ ALGORITHM

3.1 Principle of RED+

At packet-arrival a flow-lookup in a hash-table storing per-flow state-information is performed. Three counters are updated, measuring the number of bytes arrived per flow, of flows in state established or penalized and of all flows. Subsequently, RED+ determines if the packet’s flow is in state "penalized" (i.e. the flow is considered unfriendly by RED+). If this happens to be the case, WRED is executed with a lower parameter-set *(min_l, max_l, $maxp_l$)*

else WRED is executed with the higher-parameter set *(*min_h, max_h, $maxp_h$*)*. Assuming $max_h > min_h > max_l > min_l$ and existence of sufficient demand from non-penalized flows the steady-state average queue-size converges between min_h and max_h, hence the penalized flows have a drop-probability of one. If the average queue size is below max_l in case of moderate congestion RED+ additionally accommodates flows in state penalized.

In order to determine which flows should be penalized RED+ computes an approximation of the max-min fair-share of the link-bandwidth in a periodic background task. We define "one period" as the constant interval of time between subsequent calls of the background task.

The background task additionally computes a flow's state as a function of its arrival rate in comparison to the fair-share. If a flow's arrival rate in times of congestion is higher than the fair share RED+ sets the flow into state "penalized", else the flow is set to state "non-existent, new or established". The transitions between these states are explained in section 3.2.

Figure 1 summarizes the operations of RED+ at packet-arrival and gives a rough overview of the background-task:

Packet arrival:
- *Perform flow-lookup*
- *Update counters for measurement of arrival rates*
- *If the flow is in state penalized*
  - *execute WRED with lower parameter-set (*min_l, max_l, $maxp_l$*)*
- *else*
  - *execute WRED with higher parameter-set (*min_h, max_h, $maxp_h$*)*

Periodic background task:
- *Approximate max-min fair share of the link capacity*
- *Compute the state of flows*

Figure 1 *RED+ pseudo-code*

3.2 RED+ State Transitions

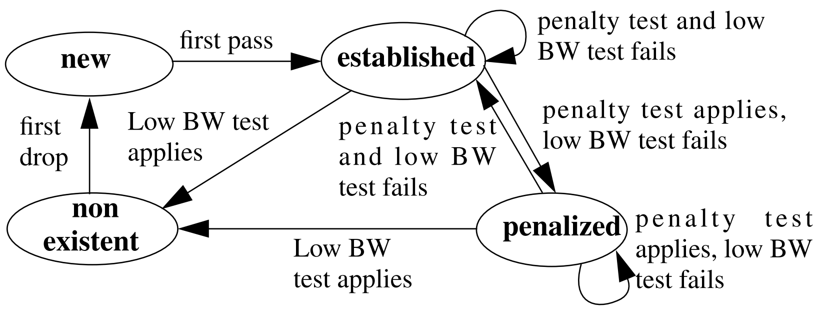

Figure 2 *RED+ state machine*

We examine the states a flow can pass during its lifetime as illustrated in figure 2:

1.) non existent: a flow is in state “non existent” if it has no per-flow-information stored. This state is virtual as RED+ is not aware of this flow.

2.) new: if a flow had its first packet drop and the hash table is not full, per flow state-information (IP Addresses, Port numbers, a counter for measurement of the per-flow arrival rate and 2 bits storing the flow’s state) is allocated. The flow’s state changes from “non-existent” to “new”. The significance of the state new is to avoid false measurement of the per-flow arrival rate which is required for the "low-bandwidth test" and the "penalty test" (see point 3). Both tests compare the number of bytes received during the last period with the fair-share. The storage of flows in the hash-table happens asynchronously to the background task. Hence, if the penalty and the low-bandwidth test were performed when the background task recognizes the flow for the first time, the arrival rate of newly stored flows would be underestimated as their byte counters had been updated for a shorter interval of time than the counters of flows already stored in the hash-table for more than one period. The meaning of the state new is to avoid the problem of underestimation of the per-flow arrival rate for flows stored during the last period. Consequently, the time between a flow’s first packet drop and storage in the hash-table and the first appliance of the penalty and low-bandwidth test is greater than one period and smaller than two periods.

3.) established: if the background task scans a flow in state new for the first time the flow’s state is changed to “established”. For established flows the arrival rate is measured and two test are performed: first, the penalty test determines if the flow’s arrival rate since the last call of the background task has been higher than the fair-share. If the penalty test applies the flow’s state is changed to "penalized". Second, the low-bandwidth test determines if the arrival rate of a flow is below the fair-share divided by the "*dealloc_param*"[3] or if this flow has the minimum arrival rate of all penalized and established flows (see section 3.3). If the low bandwidth test applies a flow’s state changes to non-existent (i.e. its per flow information in the hash-table is deallocated).

4.) penalized: For penalized flows the arrival rate is measured and two test are performed: the penalty test determines if the flow’s arrival rate since the last call of the background task has been lower than the fair-share and the flow’s state should change to established; the low-bandwidth test is applied as explained above in point 3.

---

[3] The dealloc_param is a constant set to 16 in our simulations. Setting this parameter higher decreases the probability that flow information of unfriendly flows with variable demand is falsely deallocated but increases the probability that unfriendly flows can not be stored immediately if the hash-table is filled to a high degree.

3.3 Details on the Background Task

In order to allow router vendors to restrict the amount of stored per-flow information to fit into the processor-cache, RED+ assumes a fixed, rather small size of the hash-table. This implies, that the maximum number of flows stored in the hash-table will generally be significantly smaller than the total number of flows traversing the output port. Hence we need a mechanism that only keeps the highest bandwidth flows in the hash table and accommodates new flows even if the hash-table is filled to a high degree. RED+ allocates flow-state if a flow experiences a packet-drop and the hash-table is not full. Note that allocating flow state for flows having packet-drops implies that flow state is likely to be stored for the high-bandwidth flows as a flow's drop-probability with RED is directly proportional to its arrival rate [24]. In order to enable storage of new flows when the hash-table is filled to a high degree, the background task deletes at least one flow having the minimum arrival rate of all flows stored in the hash-table in each period (see low-bandwidth test described in section 3.2). Allocating flow-state to flows likely having a high arrival rate (i.e. flows experiencing drops) and deleting flow-state of the lowest-bandwidth flows converges to a "maximum arrival rate allocation of the hash-table" (i.e. only the highest bandwidth-flows are permanently stored in the hash-table). Deleting the flows with the lowest arrival-rate additionally solves the task of deallocating flows which have stopped transmitting packets.

For computation of the max-min fair-share we use a derivate of the iterative mechanism explained in [20]. On the contrary to the mechanism in [20], RED+ only has partial knowledge of the per-flow arrival rates as flows in state non-existent and new are not taken into account. However, this lack of information can be compensated, the max-min fair-share can be computed in all relevant cases and the maximum possible number of unfriendly flows given a certain capacity of the hash-table can be penalized if the algorithm in [20] is initialized differently (see appendix and [23] for details). After initialization, RED+ proceeds with the computation of the max-min fair share like the original mechanism.

In each iteration of the algorithm computing the fair share the whole hash-table has to be scanned. Although only comparisons and additions are required, scanning the hash-table arbitrary times would mean too much overhead for a real implementation. Hence we stop after n iterations (formally speaking, we perform an *n'th* order approximation to the fair-share in the max-min sense). In all our simulations, the max-min fair-share was approximated sufficiently accurate after two iterations (see [23]).

The penalty test seems straight forward: if the arrival-rate of a flow during the last period is greater than the fair share, the flow's state is set to penalized, else the flow's state is set to established. However, by simulation (see [23]) we figured out that this policy would cause global synchronization in scenarios with TCP flows because several TCP conversations would be penalized at one instance in time. Penalizing TCP flows means enforcing a high packet drop rate. The TCP-senders in turn significantly reduce their congestion window (and thereby their sending rate) at the same point in time causing global synchronization. To avoid this problem we only penalize one flow - the established flow with the maximum arrival-rate, and un-penalize another flow - the penalized flow with the minimum arrival rate. Obviously, this implies the drawback of longer convergence times. Consider a scenario with a period of k seconds. At one point in time l unresponsive CBR flows start to send with a rate above the fair share. It will take $k * l$ seconds until all CBR flows are in state penalized.

The periodic background task is invoked in constant time intervals. We have performed simulations with periods between three and eight seconds [23]. A period of 5 seconds seems appropriate in most cases. Obviously, longer time-intervals between calls of the background task cause longer convergence times for detection of multiple unfriendly flows. Shorter time-intervals between calls of the background task cause more load for the router. The background task can be made work-conserving by elongating the time-interval between calls of the background task directly proportional to the total arrival rate at the router output-port.

For detailed explanations of RED+ pseudo-code and discussion of further properties we have to refer to [23] due to space-limitations in this paper.

4 SIMULATION OF RED+

We have implemented the RED+ algorithm in the ns simulator [10], version 2. Simulations in [23] include scenarios investigating different RED+ parameter settings, topologies, multiple congested gateways and different mixes of TCP, CBR and ON/OFF flows. Additionally, scenarios showing RED+ with ECN packet-marking instead of dropping [11][21] for flows in state non-existent, new and established, different bottleneck-link capacities, different RTTs and number of TCP flows are investigated. RED+ succeeds in detection of unfriendly flows in all scenarios. Further simulations show that the max-min fair share is sufficiently accurately approximated if the maximum number of iterations of the algorithm computing the fair-share ("n" parameter) is set to 2.

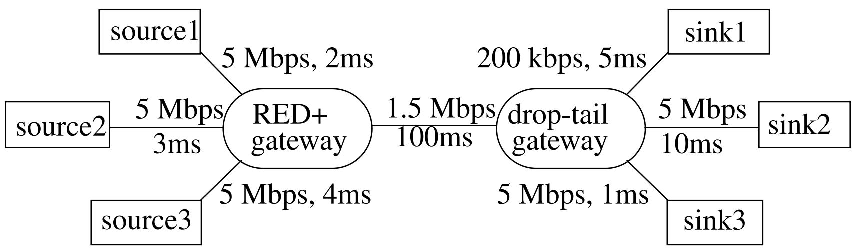

Figure 3 *Simulated network*

RED+ parameters (see [8] for an explanation of RED-specific parameters): *byte-mode* = false, w_q = 0.002, min_l = 20, max_l = 40, $maxp_l$ = 1, min_h = 50, max_h = 150 packets, $maxp_h$ = 0.1, ECN is disabled, *mean-pktsize* = 500 bytes, *period-length* = 4 sec, *dealloc-param* = 16, *n* = 2, the hash-table stores state-information of 15 flows at maximum.

Other parameters:Simulation duration: 50 seconds, buffers at router output ports store 200 packets. All output-ports except the output-port at the RED+ gateway served by the 1.5 Mbps link use drop-tail queue management.

Traffic: 3 CBR flows (flows 1,2,3) with rates of 400, 200 and 100 kbps start at 0, 5 and 10 seconds of simulation time. Flow 1 is routed from source 1 to sink1 hence it uses the 200 kbps link. 97 TCP-Reno and TCP-SACK flows transmit packets from sources to sinks and start randomly between zero and 10 seconds of simulation time. Packet sizes of TCP flows are uniformly distributed with a mean of 500 bytes. None of the flows terminates prior to the simulation.

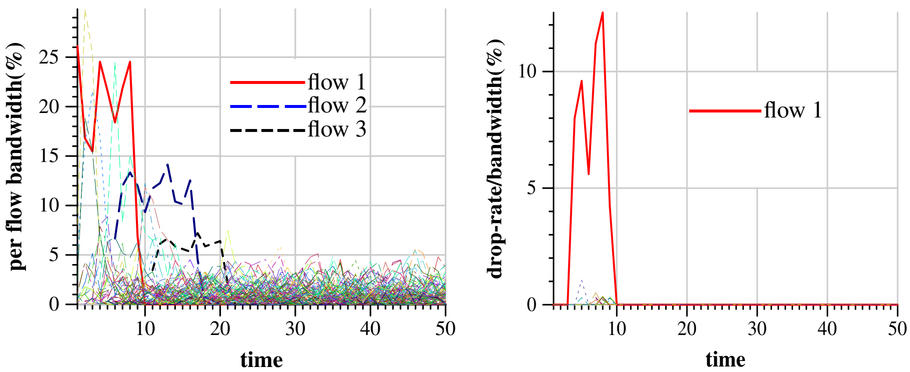

Figure 4 *Left part: per-flow bandwidth allocation as a percentage of the bottleneck-capacity between the RED+ and the drop-tail gateway; right part: per-flow droprate at the 200 kbps link as a percentage of the link-capacity between the RED+ and the drop-tail gateway.*

As shown in the left part of figure 4, RED+ sets the unresponsive CBR flows into state penalized between 10 and 20 seconds of simulation time, starting with the highest bandwidth flow. CBR flows are detected as unfriendly within 2 periods since their first packet-drop and allocation of flow-state. Due to their unresponsiveness they stay in state penalized and are shut out for the rest of the simulation.

During the first seconds of simulation-time flow1 (the CBR flow traversing the second bottleneck) is not penalized, hence flow1 consumes a significant portion of the bottleneck-link capacity between the RED+ and the drop-tail gateway and experiences vast packet drops at the second congested link between the drop-tail gateway and sink1. This behavior causes wastage of bandwidth and may cause congestion-collapse in the extreme case[4] [7]. As soon as flow1 is in state penalized its share of the link-capacity at the first bottleneck - and thereby its droprate at the second bottleneck - equals zero (see figure 4, right part).

This simulation shows that RED+ is able to penalize unfriendly flows in case the total number of flows is significantly larger than the capacity of the hash-table. The hash-table is capable of storing state-information of 15 flows while the total number of flows traversing the RED+ output-port equals 100.

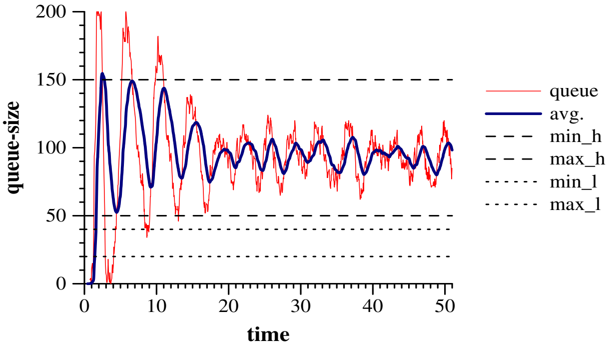

Figure 5 *Average and instantaneous queue-size over time at the RED+ gateway*

Figure 5 shows the reason why flows in state penalized are completely shut out: the average queue size converges between min_h and max_h as there is sufficient demand from CBR and TCP flows. Max_l is smaller than min_h, hence flows in state penalized experience a drop-probability of one.

5 CONCLUSIONS

As shown by simulation, RED+ is able to solve the problem of unfairness and congestion-collapse due to unresponsive best-effort flows. Unresponsive flows are completely shut out once they are detected. Due to this severe form of discrimination the current incentive for users to be misbehaving and to create unresponsive flows would disappear in case RED+ was deployed in the Internet. On the contrary, users would get the desirable incentive to implement conservative end-to-end congestion control.

While providing these functionalities RED+ only stores state-information of a small portion of the flows traversing the router output-port. Additionally, RED+ only requires a few additions (counter updates at packet arrival) besides the flow-lookup and the execution of a drop-preference mechanism (like WRED) in the data-forwarding-path. More complex operations are performed in a periodic background task which can be made work-conserving.

---

[4] If the arrival rate of flow1 was greater than or equal to the link-capacity between the RED+ and the drop-tail gateway, flow1 had a throughput of 200kbps. All other flows were shut out completely.

As opposed to [23] where the behavior of RED+ has been extensively investigated (see section 4 for a brief overview), the limited scope of this paper only allows showing a few simulations. [23] additionally shows detailed pseudo-code of the algorithm lacking in this paper.

Discrimination between friendly and unfriendly flows can not be perfect with mechanisms like RED+. Friendly TCP flows may be penalized falsely and unfriendly flows may not be penalized at all if their arrival rate in times of congestion is only marginally higher than the fair share. If a TCP flow is penalized most of its packets are dropped causing significant reduction of its arrival-rate at the RED+ gateway. As a consequence, the penalization is removed from the TCP flow in the next period. Figuring out the operational bounds of RED+ with regards to falsely penalizing TCP flows and failing to identify unfriendly flows will be the task of a future paper. Additionally, the behavior of RED+ in the presence of unfriendly but responsive flows will be investigated.

6 REFERENCES

[1] J.C.R. Bennett, H. Zhang, "Hierarchical Packet Fair Queueing Algorithms", Proc. of ACM SIGCOMM, 1996

[2] S. Blake et al., "An Architecture for Differentiated Services", RFC 2475, December 1998

[3] B. Braden, V. Jacobson et al., "Recommendations on Queue Management in the Internet", Internet draft, March 1997

[4] Cisco Web-pages, http://www.cisco.com/warp/public/732/netflow/qos_ds.html

[5] D. Clark, "Explicit Allocation of Best Effort Packet Delivery Service", http://www.ietf.org/html.charters/diffserv-charter.html

[6] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm", Proc. of ACM SIGCOMM, 1989

[7] S. Floyd, K. Fall, "Promoting the Use of End-to-End Congestion Control", Submitted to IEEE/ACM Transactions on Networking, February 1998, http://www.aciri.org/floyd

[8] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transaction on Networking, August 1993

[9] S. Floyd, V. Jacobson, "On Traffic Phase Effects in Packet Switched Gateways", Computer Communications Review, 1991

[10] NS Simulator Homepage, http://www-mash.cs.berkeley.edu/ns/

[11] S. Floyd, "TCP and Explicit Congestion Notification", http://www-nrg.ee.lbl.gov/floyd/ecn.html

[12] P. Gowal, H.M. Vin, H. Cheng, "Starttime Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks", Proc. of ACM SIGCOMM, 1996

[13] E.L. Hahne, "Round Robin Scheduling for Max-Min Fairness", JSAC, 1991

[14] V. Jacobson, "Congestion Avoidance and Control", Proc. of ACM SIGCOMM, Aug.1988

[15] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm", Message to end2end -interest mailing list, April 1990

[16] D. Lin, R. Morris, "Dynamics of Random Early Detection", ACM SIGCOMM, 1997

[17] M. Mathis et al, " The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", Computer Communications Review, July 1997

[18] McKenney, P., "Stochastic Fairness Queueing", Proc. of IEEE INFOCOM, 1990

[19] T.J. Ott, T.V. Lakshman, L.H. Wong, "SRED: Stabilized RED", IEEE INFOCOM, 1999

[20] K. K. Ramakrishnan, D. Chiu, R. Jain, "Congestion Avoidance in Computer Networks with a connectionless Network Layer; Part 4, A selective binary feedback scheme for general topologies", DEC-TR-510, 1987

[21] K.K. Ramakrishnan, S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC2491, January 1999, http://www.aciri.org/floyd/ecn.html

[22] I. Stoica, S. Shenker, H. Zhang, "Core-stateless Fair Queueing: achieving approximately

fair Bandwidth-Allocations in High-Speed Networks", Proc. of ACM SIGCOMM, 1998
[23]T. Ziegler, U. Hofmann, S. Fdida, "RED+ Gateways for detection and discrimination of unresponsive flows", Technical Report, December 1998, http://www-rp.lip6.fr/InfosTheme/Anglais/publicationsan.htm
[24]T. Ziegler, S. Fdida, U. Hofmann, "A distributed Mechanism for detection and discrimination of non-TCP-friendly Flows in the Internet", February 1999, http://www-rp.lip6.fr/InfosTheme/Anglais/publicationsan.htm

APPENDIX

[20] proposes an iterative mechanism computing the max-min fair-share in iteration-step i (A^i_{fair}) as follows:

$$A^i_{fair} = \frac{B - d^i_{low}}{h^i}$$

B denotes the link-capacity, d^i_{low} denotes the sum of the arrivalrates of the flows having arrival rates below or equal A^i_{fair}, h^i denotes the number of flows having arrival rates higher than A^i_{fair}. The d_{low} quantity is initialized to zero, h is initialized to the total number of flows. If $h^i = h^{i-1}$, A^i_{fair} equals the max-min fair share (A_{fair}) and the iteration can be terminated. A max-min fair allocation of B fully satisfies the demand of flows having arrival rates below A_{fair} and restricts flows having arrivalrates above A_{fair} to A_{fair}.

As mentioned in section 3.1, RED+ measures the per-flow arrival rate of flows in state penalized or established, the total arrival rate of flows in state penalized or established (λ_{ep}) and the total arrival rate of all flows (λ_{all}). For computation of the fair-share, RED+ initializes the d_{low} quantity to the total arrival rate of flows in state new or non existent, λ_{all} - λ_{ep}; h is initialized to the number of flows in state penalized or established. After the initialization RED+ continues with the computation of the max-min fair share as explained above for the original mechanism.

For the following considerations we assume that the process of maximum arrival rate allocation (see section 3.3, first paragraph) is in steady state, i.e. only the highest bandwidth flows are stored in the hash-table. The n parameter limiting the numbers of iterations of the algorithm computing the fair share, is assumed to be infinity.

Theorem: RED+ penalizes the maximum possible number of flows having arrival rates above the max-min fair share given a certain size of the hash-table.

Explanation: Let m denote the maximum number of flows which can be permanently stored in the hash-table. We distinguish between two cases and show that in the first case exactly the flows having arrival rates above the max-min fair share are penalized, in the second case the maximum possible number of flows above the max-min fair share (i.e. m flows) is penalized.

First case: the number of flows with arrival rates higher than the max-min fair-share is smaller than or equal m. Under the assumption of maximum arrival rate allocation, the demand of each flow in state non-existent or new has to be below the max-min fair-share in this case. The arrival rate of flows in state non-existent or new would contribute to d_{low} in the original mechanism, hence we may initialize d_{low} to the total arrival rate of flows in state new or non existent, λ_{all} - λ_{ep}. As only flows in state established or penalized remain to be considered h can be initialized to the number of flows in state penalized or established. After the algorithm has terminated, the max-min fair share computed by RED+ equals the value computed by the original mechanism with knowledge of all per-flow arrivalrates[5].

Second, inverse case: there are flows in state non-existent or new having a demand above the max-min fair share. As convergence to a maximally arrival rate allocation of the hash table has been achieved, we know that any flow in state established or penalized has a higher arrival-rate than any flow in state non-existent or new. Hence all flows in state penalized or established have to have arrivalrates above the max-min fair share either.

The total portion of the link bandwidth RED+'s derivate for computation of the max-min fair share allocates to the established and penalized flows is given by *maximum(0, B -* d_{low}*)*, where d_{low} equals the total arrival rate of the flows in state non-existent or new. From the assumption of existence of flows in state non-existent or new having an arrival rate above the max-min fair share follows that RED+'s d_{low} is greater than the d_{low} value of the original mechanism. Consequently, RED+ allocates a smaller portion of the link-bandwidth to the established and penalized flows than the original mechanism; therefore the fair-share computed by RED+ is smaller than the max-min fair-share computed by the original algorithm. As flows in state penalized or established have arrivalrates above the max-min fair share (see last paragraph) and RED+'s fair-share is smaller than the max-min fair share all flows permanently stored in the hash-table are penalized.

Although the algorithm fails in computing the max-min fair share in the second case, it is correct to penalize the flows stored in the hash-table as these flows have arrivalrates above the max-min fair-share, as shown above. Obviously, not all flows with arrival rates above the fair share are penalized as there are flows with an arrival rate above the max-min fair-share in state non-existent or new which are not taken into account by RED+. However, RED+ penalizes the maximum possible number of the highest bandwidth flows traversing the output-port, given a certain size of the hash-table.

---

[5] Note that computing the max-min fair-share does not mean that RED+ enforces a max-min allocation of the link-bandwidth. The allocation of the link-bandwidth is determined by the distinct packet drop probabilities for penalized flows and all other flows.