Int. j. adv. multidisc. res. stud. 2022; 2(2):318-321

Received: 14-02-2022
Accepted: 24-03-2022

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

Window-Based Congestion Control

[1] Matthew O Ayemowa, [2] AA Waheed, [3] OL Abraham
[1] Department of Computer Science, Gateway Polytechnic, Saapade, Ogun State, Nigeria
[2] Department of Computer Science, Lead City University, Oyo State, Nigeria
[3] ITS Unit, Gateway Polytechnic, Saapade, Ogun State, Nigeria

Corresponding Author: **Matthew O Ayemowa**

Abstract
Transmission control protocol (TCP) or Internet Protocol (IP) has undergone several transformations. There are many proposals that have been put forward to change the mechanisms of TCP congestion control to improve its performance. Much work has been done on congestion avoidance/control, mostly, previous studies on window-based congestion control were based on the starting network, adaptive congestion, end to end mechanism, however, network overloading should be prevented and packet losses should also be minimized. The aim of this research is to introduce a Proactive Explicit Rate Control (PERC) algorithm for max-min fair rates that will take care of network overloading and packet losses by sharing. We introduced a PERC algorithm for max-min fair rates that can give priority to short flows as needed.

Keywords: Congestion Window, Network Overloading, Network Protocols, Packet Losses, Congestion Control

Introduction
Window-Based Congestion Control is an internet work horse, in which, up to 90% of the traffics are managed by the Transmission Control Protocol (TCP) which are used for web browsing, file sharing, e-mail transmission and other applications. The Congestion control algorithms allows different applications to share network bandwidth efficiently without oversubscribing or overloading the links. Congestion happens when a network link is overloaded by multiple applications, the buffers fill up, packets will be dropped and retransmitted, and applications see a spike in latency or a drop in goodput. User-facing applications such as search and interactive web services care about latency, while back-end applications like e-mail or gmail backups care about goodput. There are many different ways of sharing the network bandwidth to further adapt to the application mix; these correspond to different objectives for congestion control algorithms for max-min fairness, shortest-flow-first, etc. Congestion control is very important when the network is heavily loaded, that is, when bandwidth demands are high and there are always multiple applications that may want to make use of the same links simultaneously. This is very important when the network speed or round-trip delay is high and buffers fill up quickly even before there is time to react. Consider two flows with 200µs Real Time Technology Solution (RTTs) that share a single 200Gb/s bottleneck link with 225KB of buffering. It only takes 20µs of line rate traffic to fill the buffers, whereas the servers can adjust their rates only at 200µs time-scale, which is how long it can take to measure and react to congestion. When the reaction time is long, relative to the buffer size, it may not be easy to control congestion reactively. As the RTTs get longer or the link capacity gets higher, as in a WAN, without a corresponding increase in the buffer size, this problem only gets worse [1].

Statement of the problem
The Congestion control optimizes the performance in a communication network. This optimization means, roughly, that sending rates at the data sources should be as high as possible, without overloading the network. The primary measure of network overload is packet losses; when the arrival rate at a link exceeds capacity, the corresponding queue starts to build up, and when the queue is full, packets must be discarded. The bottleneck links in the network should be fully utilized.

Aim and objectives
The aim of this research is to introduce a Proactive Explicit Rate Control (PERC) algorithm for max-min fair rates that will

take care of network overloading and packet losses.

The aim is to be achieved by the following objectives.

1. To design a new model/algorithm that will take care of *Network Overloading*.
2. Previous Authors used TCP (Transmission Control Protocol), DCTCP (Data Center Transmission Control Protocol) and RCP (Remote Copy Command), but this thesis will use PERC (Proactive Explicit Rate Control) for data center networks to apply the algorithm for *Tracking Packet Losses*.
3. To verify the performance improvements of our proposals (Simulation).

Scope of the study

- In this research, we introduced a PERC algorithm for max-min fair rates that can give priority to short flows as needed. To the best of our knowledge, s-PERC ("stateless Proactive Explicit Rate Control") is the first practical PERC algorithm that converges to exact max-min rates in a known bounded time. This algorithm does not require switches to be synchronized or to maintain per-flow state, and it can be proved to converge in at most 6*N* RTTs, where *N* is the length of the longest dependency chain.
- To the best of our knowledge, s-PERC ("stateless Proactive Explicit Rate Control") algorithm is the first practical algorithm that converges to exact max-min rates in a known bounded time.
- This algorithm does not require switches to be synchronized or to maintain per-flow state.

Significance of the study

Contributions of this work, we introduce new delay-based techniques in order to improve an existing loss-based, end-to-end, congestion control algorithm.

We believe that these techniques will bring significant performance advantages into numerous network scenarios.

We believe that such techniques can bring significant performance advantages in numerous network scenarios. We run simulations with the Network Simulator 2 (NS-2) tool to verify the performance improvements of our proposals. The simulation results focus on key performance metrics such as link efficiency, fairness, router queue occupancy, latency, and packet loss rate. Finally, we compare our proposals with several other high-speed congestion control algorithms. A TCP end-to-end congestion control algorithm is the simplest and most scalable form of congestion control in the Internet. Except for the sender, and possibly the receiver, the end-to-end congestion control approach treats all components of the Internet architecture as black-boxes. Congestion is inferred only through two implicit signals: packet losses and delay variations.

Materials and methods

We introduced a PERC algorithm for Packet Losses and Network Overloading in order to achieve the aim and objectives of this research. We stimulated s-PERC's technique of propagating bottleneck rates only when they are high enough. We also presented simulation results that indicate that n-PERC can take an indiscriminately long time to converge in the worst case.

PERC algorithm without Per-Flow State

The n-PERC algorithm that was designed eliminates per-flow state at the links. Each link (*l}* has an estimate of the set of flows bottlenecked at the link and, and this was called *B*^(*l*) and Ê(*l*) to distinguish them from the true sets. Whether a flow is in *B*^(*l*) or Ê(*l*) is carried in the control packet of the flow, for each link *l*, the link does not need to store per-flow state. Recall that in the *Fair* algorithm the link uses per-flow state to store the limit rate of every flow, that is, the rate that the flow is limited to by the rest of the network.

In the n-PERC algorithm, the link stores two summative values only, called *SumE* and *NumB*. *SumE* is the sum of the allocations of flows in Ê(*l*), where each flow is allocated exactly its limit rate, and *NumB* is the number of flows in *B*^(*l*), which are all limited at *l*. While these values of *SumE* and *NumB* may not yield the correct local max-min fair rate, they do yield some rate *R* using Equation below, replacing the actual values of SumE and NumB with *SumE* and *NumB*:

$$R = \frac{C - SumE}{NumB:}$$

This rate *R* in turn can be used to reclassify a flow into *B*^ and Ê based on its latest limit rate. The question to ask is whether the rates converge over time to the ideal max-min rates.

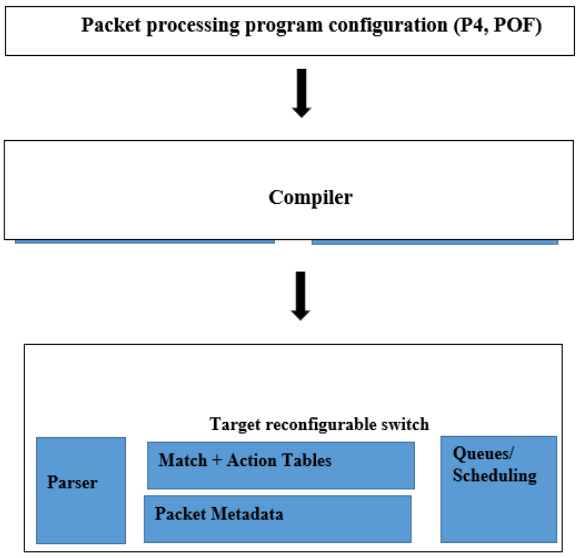
Results and discussion

Algorithm 5 (Network Overloading) Timeout action at link *l* for s-PERC, every round starting from time $T_0(l)$

1: *SumE* = 0: sum of allocations of flows bottlenecked elsewhere, that is, in Ê 2: *NumB* = 0: number of flows bottlenecked here, that is, in *B*^	►Initial state at link *l*
3: *MaxE* = 0: maximum allocation of flows moved to Ê since last round	
4: $MaxE^1$ = 0: maximum allocation of flows moved to Ê in this round	
5: if *s*[*l*] = E then	►Flow was last bottlenecked elsewhere
6: *SumE* ← *SumE-a*[*f*] 7: *NumB* ← *NumB* + 1	► Update link state to assume flow is going to be bottlenecked
8: $b \leftarrow \frac{C\text{-}SumE}{NumB}$	
9: *e* ← $min_{m2Pf\ nl}$; i[*m*]=0*g b*[*m*] (or *1* if there is no other link in *Pf* with ignore bit unset)	►Find flow's new limit rate
10: *a* ← min (*b; e*).	►Find flow's new allocation
11: **if** *b* <= *e* **then** *s*← B.	►Flow is now bottlenecked here, classify as *B*
12: **else** *s* ← E.	►Flow is now bottlenecked elsewhere, classify as *Ê*
13: *b*[*l*]← *b; a*[*l*] ← *a; s*[*l*] ← *s*	►Update control packet
14: **if** *b* < *MaxE* **then** *i*[*l*] ← 1	►Bottleneck rate is low; do *not* propagate it
15: **else** *i*[*l*]← 0.	►Bottleneck rate is high enough; propagate it

16: **if** flow is leaving **then**.	►Update link state to remove flow
17: *NumB* ← *NumB* – 1	
18: **else if** *s* = E **then**.	►Update link state to reflect flow is in $\hat{E}$
19: *NumB* ← *NumB* – 1	
20: *SumE* ← *SumE* + *a*	
21: *MaxE* ← max (*MaxE; a*)	
22: $MaxE^1$ ← max ($MaxE^1$, *a*)	

Algorithm for Control Packet Processing at link *l* for s-PERC



Network Overloading

Algorithm 7 The WFk algorithm to compute max-min rates. For any link *l* that is removed in iteration *n*, we allocate *A*[*f*] = *R*[*l*] = *C*[*l*]=*N*[*l*] as computed in iteration *n* to each of its flows in *Ql* during iteration *n*, and remove the flow.

1. *L*: set of all links in the network that have at least one flow.
2. *C*: remaining link capacities, *N*: number of flows
3. *R*: remaining link capacity per flow, *Q*: active flows
4. *A*: bandwidth allocation to flow, *P*: array of links per flow
5. *iteration* ← 0
6. **while** *L* is not empty **do**
7. *iteration* ← *iteration* + 1
8. **for all** *l 2 L* **do** *R*[*l*] ← *C*[*l*]=*N*[*l*]
9. *links_to_remove* ← *fg; flows_to_remove fg*
10. *f***or all** *l Є L* **do**
11. **if** WF^k == WF∞ **then** *minRate* = ←min*x*∞*L R*[*x*]
12. **else** *minRate* = *mini*=1 min*x*∞ *Xi*(*l*)[*R*[*x*]
13. **if** *R*[*l*] == *minRate* **then**
14. add *l* to *links_to_remove*
15. **for all** *f Є Q*[*l*] **do**
16. add *f* to *flows to_remove*
17. *A*[*f*] = *R*[*l*]
18. **for all** *f 2 flows_to_remove* **do**
19. **for all** *l Є P*[*f*] **do**
20. *C*[*l*] ← *C*[*l*] - *A*[*f*]
21. *N*[*l*] ← ~~*N*~~[*l*] – 1
22. remove *f* from *Q*[*l*]
23. remove *links_to_remove* from *L*

Conclusion

Most of existing congestion control algorithms are essentially reactive control systems, which will figure out rates purely by reacting to congestion signals typically at RTT time scales, then taking small steps over many iterations toward convergence. As networks get faster and more data can fit into each RTT, buffers can fill up quickly even before there is time to react. Hence, in this research, we focused on a different class of algorithms: PERC algorithms, which do not rely on congestion signals but instead use explicit global information (like the number of flows crossing a link) to proactively compute rates for all flows. We believe that congestion control should converge in a time limited only by fundamental dependency chains, which are a property of the traffic matrix and the network topology. Prior attempts to proactively calculate the fair share rates in the network were not successful, because they required per-flow state, and the algorithms were not proved to converge.

With s-PERC, we have introduced a PERC algorithm that is practical (it does not require per-flow state, and the calculations are also possible at line rate in relatively simple hardware) and guaranteed to converge; our results from the simulations and a hardware prototype show that s-PERC is robust to churn and to also converges several times faster than other algorithms. We have evaluated PERC in a data-center setting to validate that PERC achieves flow completion times that are closer to an ideal max-min scheme than a reactive algorithm, because of its faster convergence. For realistic workloads, PERC competes favorably with schemes that favor short flows, such as p-Fabric, yielding low latencies for short flows and high throughput for large flows. The simulations indicate that fast-converging PERC algorithms like s-PERC would be very well suited to long-haul networks where there is a need for fair bandwidth allocation between flows that that is predictable and avoids congestion. As we have mentioned in this chapter, there is more work to be done, and so we believe that this is only the first word about PERC algorithms in high-speed networks, not the last.

References

1. Niels Moller. Window-Based Congestion Control a Dissertation Submitted to Stockholm, Sweden, 2018. ISBN-978-91-7178-831-3.
2. Rhee, L Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In International Workshop on Protocols for Fast Long-Distance Networks, Lyon, February 2015.
3. M¨oller N, Johansson KH, Hjalmarsson H. Making retransmission delays in wireless links friendlier to TCP. In IEEE Conference on Decision and Control. IEEE, 2014.
4. Niels Moller. Window-Based Congestion Control a Dissertation Submitted to Stockholm, Sweden, 2018. ISBN-978-91-7178-831-3.
5. Kanagarathinam MR, Singh S, Sandeep I, Kim H,

Maheshwari MK, Hwang J, *et al*. NexGen D-TCP: Next Generation Dynamic TCP Congestion Control Algorithm. IEEE Access. 2020; 8:164482-164496.
6. Patil J, Tokekar V, Rajan A, Rawat A. SMDMTS–Scalable Model to Detect and Mitigate Slow/Fast TCP-SYN Flood Attack in Software Defined Network. In 2020 International Conference on Computational Performance Evaluation (ComPE). IEEE, 2020, 290-295.
7. Guan S, Jiang Y, Guan Q. Improvement of TCP Vegas algorithm based on forward direction delay. International Journal of Web Engineering and Technology. 2020; 15(1):81-95.
8. Ahmad M, Ahmad U, Ngadi MA, Habib MA, Khalid S, Ashraf R. Loss Based Congestion Control Module for Health Centers Deployed by Using Advanced IoT Based SDN Communication Networks. International Journal of Parallel Programming. 2020; 48(2):213-243.
9. Zhu J, Jiang X, Jin G, Li P. CaaS: Enabling Congestion Control as a Service to Optimize WAN Data Transfer. In International Conference on Security and Privacy in Digital Economy. Springer, Singapore, 2020, 79-90.
10. Li Y, Cao G, Wang T, Cui Q, Wang B. A novel local region-based active contour model for image segmentation using Bayes theorem. Information Sciences. 2020; 506:443-456.