

Hierarchical Congestion Control (HCC): Cooperation of Uncorrelated Flows for Better Fairness and Throughput

Shiva Ketabi, Yashar Ganjali

Department of Computer Science, University of Toronto

Abstract—Congestion control protocols face several challenges for achieving max-min fairness and high throughput. First, each flow has a limited view of the network state. In the absence of a centralized congestion control entity, coordination is left (directly or indirectly) to individual flow sources. Second, most flows are very volatile by nature: flow rates/demands change significantly from one instant to another.

In this paper, we present a hierarchical congestion control scheme to tackle these challenges. We aggregate flows with low correlation in a hierarchical manner, and recursively compute and allocate rates to these flows. Our experiments show that correlation-aware aggregation results in significant improvements in terms of fairness (around 15% increase) and throughput (around 100% increase) compared to schemes that neglect the correlation metric. Also, we evaluate our rate allocation scheme and show that it can achieve near optimal rates for 95th percentile of flows in most time intervals.

I. INTRODUCTION

A common weakness of most congestion control protocols is lack of cooperation and information sharing among flows. Congestion control protocols that are distributed by design, including traditional ([1]–[3]) or more recent protocols ([4]–[8]), delegate the task of congestion detection to individual flows. These individual flows are required to estimate the congestion state based on limited congestion signals they receive. These signals, ranging from packet loss (*e.g.* TCP Reno [1]) to variations in packet round-trip latencies, (*e.g.* TCP Vegas [3]) are limited by nature given the limited information provided (single or a few bits of information).

This limited view of individual flows and lack of coordination among flows make reaching optimal state challenging. In particular, it is extremely difficult to reach optimum fairness and throughput in short periods of time, especially for short-lived flows. In contrast, let us imagine a congestion control framework which has access to all congestion control signals in the network, and can use this information to accurately infer congestion state of individual flows. This, for example, can happen if we have a logically centralized controller ([9]–[12]) in the network that collects all individual congestion signals and globally allocates packet injection rates to individual flows, (*e.g.* by solving an optimization problem aiming at maximizing throughput while respecting fairness). Unfortunately, such a centralized congestion controller has

its own scalability challenges of gathering and processing information from a large number of flows in a timely manner.

A novel approach in this direction is RackCC [13], which uses tunneling to aggregate all flows between any given pair of top of rack switches (*i.e.* ToR-ToR traffic) as a single jumbo-flow. Rack-level flow aggregation helps flows to combine their individual limited knowledge of the congestion state and to collectively agree on their optimum rates. Moreover, new flows, with no prior knowledge of the network, can acquire the congestion state of older flows and immediately set a near-optimal sending rate instead of wasting time and resources searching for the optimum rate.

In this paper, we propose Hierarchical Congestion Control (HCC), a model for hierarchical aggregation of congestion signals among flows sharing bottlenecks in the network. HCC is a generalization of RackCC which allows arbitrary aggregation of flows rather than limited ToR-ToR flow aggregation.

In addition to being more general, HCC eliminates a fundamental problem in RackCC: flows that share the same source and destination racks are usually correlated. This is due to the locality of applications (*i.e.* to enhance latency) and due to the fact that these flows usually receive similar congestion signals from the network. Instead, HCC aggregates flows based on their correlation: the lower the correlation of any pair of flows, the higher their chance of being aggregated in the same group. Intuitively, the central limit theorem [14] suggests that by aggregating an adequately large number of independent flows, we will have a smooth and predictable distribution of demand rates, as opposed to significant fluctuations in aggregate demands of correlated flows.

Our proposed solution has two main phases: flow aggregation and rate allocation. For flow aggregation, we group flows that *i)* share a bottleneck on their path, and *ii)* are uncorrelated according to a correlation metric that will be introduced in Section II. A group of flows creates a super-flow, which itself is an input to the next iteration of the flow aggregation phase. The result of the flow aggregation phase after a number of iterations is a hierarchy of flows and super-flows that can be used for rate allocation.

Given the hierarchical flow groups, we assign an agent to each flow/super-flow that has two responsibilities for rate allocation. First, each agent receives updates about the allocated share from its parents in the hierarchy and accordingly

calculates the effects on the fair share of its children. Second, it gathers updates from its children about congestion signals or demands and propagates the aggregated changes to its parents.

In Section II, we explain the flow aggregation and rate allocation phases in more details. Section III presents experimental results that *i)* show the gains of uncorrelated grouping of flows in terms of fairness and throughput, and *ii)* the accuracy of the rate allocation phase and how close it is to the optimal centralized algorithms.

II. HIERARCHICAL CONGESTION CONTROL MODEL

HCC consists of a *flow aggregation* phase running every T_{agg} seconds, and a *rate allocation* phase running every T_{alloc} seconds. We assume $T_{\text{alloc}} \ll T_{\text{agg}}$. The flow aggregation phase groups flows recursively to create a hierarchy of flows used in the rate allocation phase. In the rate allocation phase, the flows in the hierarchy cooperate to find their fair share and distribute their information about the congestion and flow demands. We explain these two phases in details next.

A. Flow Aggregation

The objective of flow aggregation phase is to produce a flow aggregation graph by iteratively merging flows in the network. We start with the initial set of all flows $F^0 = \{f_1^0, f_2^0, \dots, f_N^0\}$ as input, generating a leaf node for each flow in the aggregation graph. During the i -th iteration, we merge flows sharing a path whose pairwise correlation is less than a threshold θ_i , calculated using Pearson Correlation Coefficient [14].

The flows in each group $f_{j_1}^i, f_{j_2}^i, \dots, f_{j_M}^i$, are replaced with a single flow f_j^{i+1} in the next iteration. The node representing f_j^{i+1} in the aggregation graph is the parent of all flows it replaces. For each flow f_j^i , we keep track of its source, destination, and a path between the source and destination, which we represent with (s_j^i, d_j^i, p_j^i) . For the merged flow f_j^{i+1} , the associated path is the path shared amongst all of its children, and the first and last node of this path are considered to be the source and destination of this flow.

The aggregation phase is finished when no more flows can be aggregated, or when i reaches a predefined threshold i_{max} . In the following two cases, we explain each of these termination conditions.

Case 1. Let us assume the aggregation is terminated at the k -th iteration since there are no shared links among flows $f_1^k, f_2^k, \dots, f_{N_k}^k$. Here, each congestion bottleneck can belong to only one of these k -th level flows. Therefore, we can assign the total bandwidth of the bottleneck to the corresponding flow, and then recursively share the bandwidth amongst the children of this flow at level $k - 1$. Since each flow in level $k - 1$ might be included in more than one flow in level k , *i.e.* it might have more than one parent, it picks the minimum assigned rate among all its parents as its rate. Each node then shares its rate with its parents so that any excess bandwidth can be redistributed to other flows which need it.

Case 2. If the iterations stop with $i = i_{\text{max}}$, we can use any known congestion control algorithm to allocate flow

rates to the super-flows generated by the flow aggregation phase. In other words, the set of flows in this iteration, *i.e.* $F^i = \{f_1^i, f_2^i, \dots, f_{N_i}^i\}$, can be considered as the only flows in the network and we can run any known congestion control scheme to allocate rates to them.

RackCC [13] is an example here where $i_{\text{max}} = 1$. After one level of aggregation, RackCC uses known congestion control mechanisms to allocate rates to these ToR-ToR jumbo-flows. In RackCC, all flows sharing a source and destination rack are grouped in one step which create jumbo-flows (F^1 in our model). Flows within a jumbo-flow share their knowledge about the congestion in the core links and decide on their sending rates together. When new flows are initiated, the established flows let them know about the shared knowledge and help them quickly reach to the optimal rate instead of wasting several RTTs trying to find the optimal rate.

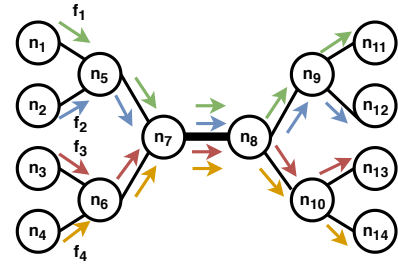


Fig. 1. An example topology.

Example. Let us consider Figure 1 as an example. We have a bottleneck link (n_7, n_8) , and four flows f_1, f_2, f_3 , and f_4 . Let us compare the following two different groupings:

Grouping 1. Merge f_1^0 and f_2^0 sharing the path (n_5, n_7, n_8, n_9) and create $f_1^1 = \{f_1^0, f_2^0\}$. Merge f_3^0 and f_4^0 sharing the path (n_6, n_7, n_8, n_{10}) and create $f_2^1 = \{f_3^0, f_4^0\}$. Note that this type of grouping does not follow our proposed approach since it aggregates flows from the same rack which are most likely correlated. In the next step, merge f_1^1 and f_2^1 since they share the path (n_7, n_8) and create $f_1^2 = \{f_1^1, f_2^1\}$. Figure 2(a) shows the resulting aggregation graph.

Grouping 2. Merge f_1^0 and f_3^0 sharing the path (n_7, n_8) and create $f_1^1 = \{f_1^0, f_3^0\}$. Merge f_2^0 and f_4^0 sharing the path (n_7, n_8) and create $f_2^1 = \{f_2^0, f_4^0\}$. Note that here we are grouping uncorrelated flows. In the next step, merge f_1^1 and f_2^1 sharing the path (n_7, n_8) and create $f_1^2 = \{f_1^1, f_2^1\}$. Figure 2(b) shows the resulting aggregation graph.

B. Rate Allocation

The rate allocation phase consists of two different rounds: congestion signal collection and congestion reaction. These rounds are performed based on the child-parent relations between flows in the aggregation graph. In the congestion signal collection round, each parent receives congestion signals from all its children, aggregates them, and sends the aggregated congestion signal to its parents. In the congestion reaction round, the action taken by each flow in response to the congestion signals is pushed from each parent to its children.

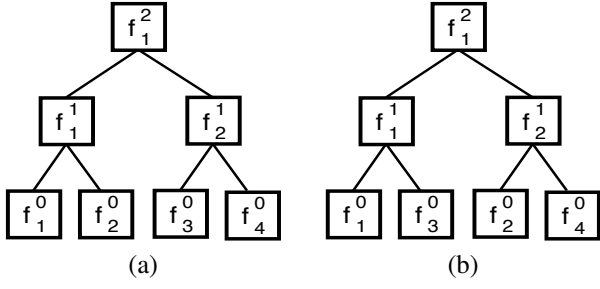


Fig. 2. Aggregation graphs. (a) correlated grouping, (b) uncorrelated grouping.

Depending on the termination case explained above, the congestion signals sent bottom-up and the actions forwarded top-down can be explicit or implicit. For the first case, in which max-min fairness is directly implemented, explicit notifications are used (*i.e.* demands and injection rates). For the second case, implicit notifications are used (*e.g.* RackCC uses ECN bits as congestion signals).

Implementation details. In our implementation, we focus on explicit rate allocation based on max-min fairness. Max-min fairness is the gold standard for rate allocation that guarantees maximum throughput while ensuring fairness in the system. Unfortunately, max-min fairness requires gathering all the input flow demands and distributing all the output allocations, which means a centralized solution based on max-min fairness is not scalable. Hierarchical Congestion Control model is an alternative distributed solution which is intrinsically more scalable.

We also use rate quantization to significantly reduce the run-time of max-min fairness. The idea is to assign flow rates/demands to certain bins and use a single value to represent all flows demands in that bin. Rate quantization has been shown to significantly reduce the run-time of max-min fairness solutions if we are willing to accept some inaccuracy [15]. This is a reasonable trade-off given the natural inaccuracy in measuring rates and time-delays associated with such measurements.

HCC maps each node in the aggregation graph resulted from flow aggregation phase to a logical agent. This agent is responsible for frequently running congestion signal collection and congestion reaction. The frequency that each agent runs these phases depends on the location of its associated flow in the flow aggregation graph. Agents closer to leaves of the hierarchy require to be updated more frequently for two reasons. First, more changes happen for flows closer to the leaves since they are the aggregation of fewer number of flows. As we move up in the hierarchy, nodes represent an aggregation of more and more flows, which are uncorrelated by design. As a result, the aggregated demands of these super-flows show less variations according to central limit theorem, and therefore need to be updated with a lower frequency. Second, agents higher in the hierarchy are usually physically associated with bottlenecks in the core of the network, thus,

they control a large number of flows. By keeping update frequency of these nodes low (compared to nodes that are closer to the leaves) we can ensure a high performance solution as collecting congestion signal and rate allocation for these nodes have a higher cost.

Needless to say, the mapping between logical agents to physical locations in a network affects the latency and the performance of our system. This is an interesting problem, which we leave as a future work. Here, we focus on the details of our rate allocation phase and how it enables a scalable max-min fairness solution.

For rate allocation phase, each agent, A_j^i , stores two lists of K numbers, where i is the height of the agent in the aggregation graph, j is the index of the agent amongst flows of the same height, and K is the number of quantization bins as we previously mentioned.

The first list, called $Demands_j^i$, is the agent's current view of the demands of all its descendant flows. More specifically, $Demands_j^i$ represents K bins, and the value of the l^{th} element in $Demands_j^i$, indicates the number of flows in the descendants of A_j^i whose demand rate equals the value of l^{th} quantization bin.

The second list is the current estimation of Agent A_j^i of the fair allocations of its descendant flows, which we denote by $Allocations_j^i$. The l^{th} element of this list indicates the fair share of the descendant flows with demand rates lying in the l^{th} quantization bin.

For rate allocation phase, each agent A_j^i updates its lists *i.e.* $Demands_j^i$ and $Allocations_j^i$ with a predefined frequency. As we discussed before, the higher A_j^i is in the hierarchy, the lower the update frequency.

Updates to $Demands_j^i$ are performed as follows.

- 1) The demand rate of a given flow changes from a quantization bin l to another quantization bin m . To represent this change, we put a -1 in the l^{th} location and a $+1$ in the m^{th} .
- 2) A new flow is initiated in a quantization bin l . For this change, we put a $+1$ in the l^{th} location.
- 3) A flow lying in quantization bin l is terminated. In this case, we put a -1 in the l^{th} location.

Each agent receives updates in demands list of all its children and sums the updates with its current demands list. If the aggregated updates is non-zero, the agent will recalculate the max-min fair share of its children according to its current share and immediately send them update messages. Besides, it sends notification messages to its own parents. If based on these notifications, the parents (or any of the ancestors) decide to change rate allocations, they will later notify this agent, which will subsequently pass the updated allocations to its children.

In response to any change in rate demands, our scheme ensures all agents optimize their allocations quickly based on their local knowledge of demands and rates. This local optimization, helps flows to react quickly to changes. The local decision may not be 100% optimal, but it helps keeping near

optimal rates while waiting for the global optimal decisions. In parallel, the agents send updates through the hierarchy and wait for global optimizations. When they receive updates from any of the parents, the agents change their allocations accordingly. This hybrid approach of local and global optimizations is one of the main contributions of our work, and it is the reason HCC leads to improved throughput and fairness in short-term using the local optimization as well as high throughput for long-lived flows.

III. EVALUATION

We implemented HCC in ns-2, and created a scenario similar to Figure 1 with 100 flows (instead of 4 flows). The simulated traffic followed an exponential on-off distribution. The simulations were run for 500 seconds, measuring fairness of each flow (using Jain index [16]) and total throughput in each second.

Figure 3(a) shows the CDF of fairness over time for correlated grouping and uncorrelated grouping. When uncorrelated flows are grouped, 50% of flows have Jain index higher than 0.75 as opposed to 0.6 for correlated grouping (Jain index is between 0 and 1 and the higher the Jain index, the more fair the algorithm is). Figure 3(b) compares the CDF of throughput of the two groupings over time and shows that grouping uncorrelated flows leads to higher throughput in general.

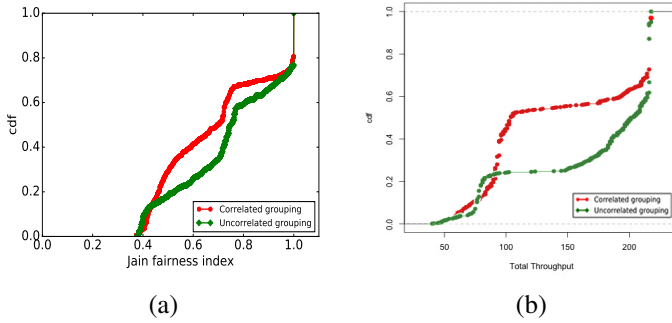


Fig. 3. Comparison of correlated and uncorrelated groupings, (a) fairness, (b) throughput.

To evaluate our rate allocation schemes (implemented in Python), we ran simulations on the same topology as in Figure 1 with 100 flows coming from 10 different racks (10 flows per rack). Similar to previous simulations on flow-aggregation, we created exponential on-off traffic: different racks had different mean values in this case. We also experimented with different values for rate allocation frequencies associated with various agents to understand the impact on accuracy and run-time of rate allocations.

We ran the experiment for 5000 intervals. In each time interval, we measured the relative distortion (error) of each flow rate compared to its optimal max-min fair rate. We calculated the 50th and 95th percentiles of these errors for different flows in each interval and plotted their CDF over time.

Figure 4 shows the impact of rate-allocation frequencies on 50th and 95th percentiles of rate distortion. Each curve in this figure corresponds to a set of frequencies identified in the legend. For example, the curve identified by [1, 2, 4] represents a system in which the frequency of updates are as follows: the agent on the top of the aggregation graph in Figure 2 updates every 4 intervals, the agents in the middle level update every 2 intervals, and the agents located at the leaves update every interval. As shown, higher frequencies result in more accurate flow rate allocations with the cost of more computation power.

Figure 4(a) shows that 50% of flows are within 10% of their optimal max-min fair rate allocation during all time intervals. Interestingly, even looking at the 95th percentiles in Figure 4(b), 95% of relative flow distortions are less than 0.2 in more than 60% of time intervals, and less than 0.4 of their rate all across the board.

These preliminary results show that HCC can achieve rate allocations close to max-min fairness while removing the need for a centralized rate allocation entity. Each distributed agent deals with a number of flows which is orders of magnitude lower than the total number of flows in the network (the case when we have a centralized rate-allocation entity). These agents aggregate and compress received signals before propagating them to layers above/below. Consequently, HCC dramatically improves the scalability of explicit rate allocation, compared to a centralized max-min fairness scheme.

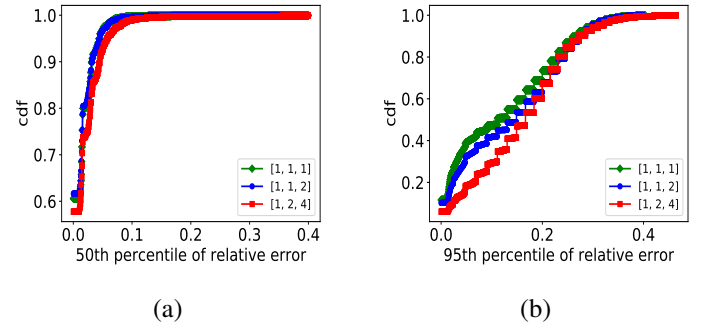


Fig. 4. CDF of relative error over time, (a) 50th percentile of flows, (b) 95th percentile of flows.

IV. CONCLUSION

Our hierarchical congestion control design makes the overhead of coordination of flows manageable, as we only need to deal with aggregated flows rather than individuals. Also, since we aggregate flows with low correlation, we are able to limit the variations in the aggregate demand (central limit theorem). Coordination among aggregate flows is also easier due to slower changes in aggregate flow demands.

An interesting problem left as a future work is how to map the logical distributed agents in HCC to physical nodes in the network. This mapping affects the latency of signaling among agents. Also, our experiments have been limited to a tree topology. We leave a comprehensive study of the proposed solution in other environments as a future work.

REFERENCES

- [1] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM computer communication review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.
- [2] —, "Modified tcp congestion control and avoidance algorithms. end2end-interest mailing list," 1990.
- [3] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465–1480, 1995.
- [4] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance," in *NSDI*, vol. 1, no. 2.3, 2015, p. 2.
- [5] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "Pcc vivace: Online-learning congestion control," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX Association, 2018, pp. 343–356.
- [6] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats *et al.*, "Timely: Rtt-based congestion control for the datacenter," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 537–550.
- [7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *ACM SIGCOMM computer communication review*, vol. 41, no. 4, pp. 63–74, 2011.
- [8] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," 2016.
- [9] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2015.
- [10] K. Winstein and H. Balakrishnan, "Tcp ex machina: computer-generated congestion control," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 123–134.
- [11] M. Schapira and K. Winstein, "Congestion-control throwdown," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 122–128.
- [12] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, "Rethinking end-to-end congestion control in software-defined networks," in *Proceedings of the 11th ACM Workshop on Hot Topics in networks*. ACM, 2012, pp. 61–66.
- [13] D. Zhuo, Q. Zhang, V. Liu, A. Krishnamurthy, and T. Anderson, "Rack-level congestion control," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 148–154.
- [14] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 1125–1133.
- [15] S. Ketabi and Y. Ganjali, "Perfect is the enemy of good: Lloyd-max quantization for rate allocation in congestion control plane," in *Proceedings of the 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019.
- [16] R. Jain, A. Durrezi, and G. Babic, "Throughput fairness index: An explanation," in *ATM Forum contribution*, vol. 99, no. 45, 1999.