New Flow Control Paradigm for Next Generation Networks[1]

Jian Pu and Mounir Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Hong Kong, China
{pujian, hamdi}@cs.ust.hk

Abstract **Quick Flow Control Protocol (QFCP) is a new congestion control protocol designed for high bandwidth-delay product networks. QFCP has two good features: "quick start" and "quick convergence". It allows a sender to start with a high initial sending rate identified by routers along the path, and all running flows converge to the fair-share sending rate quickly based on feedback from routers. Although it needs the assistance of routers, QFCP does not require routers to store any per-flow information or to do any complex per-packet calculation. The rate allocation algorithm is quite simple and only needs to be run periodically by routers. We have implemented QFCP in Network Simulator (NS). Simulations have been done to address the issues such as fairness, convergence, responsiveness and utilization, as well as flow completion time for Poison-arriving Pareto-distributed-size flows. Performance evaluation of these simulations is presented in this paper. The preliminary results are promising.**

I. Introduction

Previous research on the Internet traffic has revealed an important feature at the flow level: most of the flows are very short, while a small number of long flows account for a large portion of the traffic [1], [2]. This is known as the heavy-tailed distribution. But when the congestion control algorithm for TCP [3] was designed, flows are modeled as "long-lived flows" instead, which means that all flows are assumed to be long enough to reach their fair-share sending rates before finishing. This assumption is acceptable when the Internet is mainly composed of low-speed links. However, if most flows are short as in today's high-speed networks, can the congestion control algorithm built in TCP continue to work efficiently?

The answer is no. On the one hand, a short flow always starts with a very low initial sending rate and often gets less than its fair-share rate before finishing. And the duration of a short flow is often significantly prolonged due to packet loss, which causes timeout and packet retransmission [4]. On the other hand, a large number of short flows may also adversely impact the performance of long flows. [5] shows that randomly generated sequence of short flows may reduce the throughput of long flows up to 10%, and some special pattern of short flows can even cause greater reduction (>85%). The reason is that short flows spend most of their lifetime in the Slow Start phase when their congestion windows increase exponentially. Thus, a burst of short flows can rapidly capture a great portion of the bandwidth and driven long flows into timeout and window-halving. But the AIMD algorithm grabs the available bandwidth very slowly and makes the time of converging to the fair-share rate very long for the long flows.

As the Internet keeps evolving into a high bandwidth-delay product (BDP) network, more and more flows are becoming short flows. And the adverse impact of TCP congestion control may become increasingly severe. We need to design a new congestion control protocol for the high-speed networks achieving the following metrics:

- For short flows, they can get high initial sending rates at the start so that they can finish quickly ("quick start").
- For long flows, they can converge to the fair-share sending rate quickly and have maintainable high throughput ("quick converge").

In most currently proposed congestion control protocols, each new flow starts with a low sending rate and then probes the network for the unused bandwidth. They only show that they can achieve fair rate allocation for long-lived flows. But short flows may suffer.

If the router can find a way to compute the number of active flows and assign the fair-share rate to each flow, there will be no need to let flows wait for many RTTs to probe the fair rate by themselves. Hence, we try to design a router-assisted congestion control protocol similar to XCP [6] and RCP [7]. The sending rate of each flow is controlled by routers along the path. The router senses the degree of congestion periodically and calculates a global fair rate for all flows passing through it. A new flow will start with the same sending rate as the other ongoing flows because they get the same feedback from routers. Additionally we want the routers:

- Don't store any per-flow information.
- Don't maintain any large hash tables or do any hash computation (vs. hash-based flow counting algorithm).
- Don't do complex per-packet calculation.

II. Protocol Design

A. Framework

We use a similar framework of Quick-Start [8], but we extend the rate-request-and-grant mechanism to the whole lifetime of a flow: (1) The sender includes a "Rate Request"

[1] This research was supported in part by the Hong Kong Research Grant Council under Grant RGC HKUST6260/04E.

field in the header of each outgoing packet and sets the initial value of this field to be the desired sending rate of this sender; (2) When the packet reaches a router, the router compares the value in "Rate Request" field with the router's own fair-share rate and puts the smaller one back into that field; (3) On receiving the packet, the receiver copies the "Rate Request" field into the "Rate Feedback" field of the corresponding ACK packet and sends it back to the sender; (4) When the sender receives the ACK packet, it reads the value in the "Rate Feedback" field and adjusts its sending rate accordingly.

The sender sets the size of its congestion window according to the rate feedback using the formula:

$$cwnd = rate * RTT, \tag{1}$$

where *cwnd* is the congestion window size, *rate* is the rate feedback, and *RTT* is the moving average Round-Trip Time measured by the sender.

B. *Rate Allocation Algorithm*

The rate allocation algorithm is used by the router to compute the fair-share rate *R* periodically. One router maintains only one global fair-share rate for each output interface. This rate *R* is the maximum allowed rate for flows going through this interface during the current control period *T*. *T* is set to be the moving average of RTTs of all packets. The current number of flows through this interface is estimated using the aggregate input traffic rate and the fair-share rate assigned in the last control period. And the fair-share rate is updated based on the flow number estimation as follows.

$$N(t) = \frac{y(t)}{R(t-T)}, \tag{2}$$

$$R(t) = \frac{C - \beta \cdot \frac{q(t)}{T}}{N(t)}, \tag{3}$$

where *N(t)* is the estimation of the flow number, *y(t)* is the input traffic rate during the last control period, *R(t)* is the fair-share rate, *C* is the bandwidth capacity of the output link, *q(t)* is the queue size, β is a constant parameter, *T* is the control period.

This parameter β can be set as a policy by the router administer. A large value of β means one wants to drain up the queue more aggressively. The theoretical analysis of the impact of this parameter is left to future work. Currently we set the value of β as 0.5 in our NS implementation.

C. *Technical Details*

Burstiness Control: When designing congestion control protocols for large BDP networks, researchers often find that pure window-based rate control is not enough and additional burstiness control is needed (e.g., FAST [9]). This is because window control is too rough and may trigger a large burst of packets injected into the network all at once (e.g., in the Slow Start phase of TCP). But such violent increase of congestion window is sometimes unavoidable in order to achieve good responsiveness and high throughput, especially in large BDP networks. In QFCP, we use the rate-based pacing to avoid possible burstiness caused by a sudden increase of congestion window. So although the sender may set a large window as approved by the rate feedback in the first ACK packet, it still needs to pace these packets out in one RTT based on the assigned sending rate. Thus, congestion window controls how many packets can be sent in one RTT, while burstiness control paces these packets out in a smooth way.

Rate Stabilization: If the assigned fair-share rate changes too quickly, the formula (2) we use to estimate the number of flows based on the previously assigned rate may fail. The reason is that there may be flows with RTT longer than the control period *T* using *R(t–2T)* instead of *R(t–T)* as the sending rate. So for the sake of the accuracy of the flow number estimation, we don't want the rate assigned for two consecutive intervals to be very different. Thus, in order to stabilize the rate, we use the average of the current computed rate and the most recently assigned rate as the new rate:

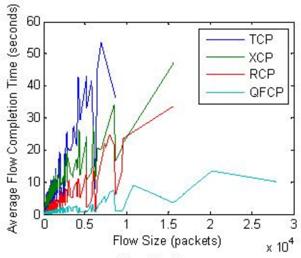
$$R(t) = \frac{R(t) + R(t-T)}{2}. \tag{4}$$

Reaction to Packet Loss: If the queue of an interface is overflowed, the router will drop packets and add the number of dropped packet to *q(t)* as *q(t)* in formula (3), because this is the real queue size that should be drained during the next interval. The router will use this "virtual queue size" in formula (3) to compute the new rate. The sender will just retransmit the dropped packets without any further reaction. This is very different from the loss-based congestion control protocols, which will adjust the congestion window if encountering packet loss. So in QFCP, it is very easy to differentiate the two kinds of packet loss: one is due to the congestion; the other is due to transmission error. Because the rate is totally determined by the routers, the routers will adjust the rate according to the degree of congestion. There is no need for the end-systems to guess whether congestion happens or not when they encounter packet loss.

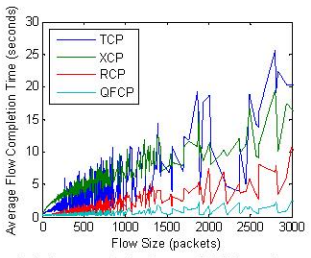
III. Simulation and Evaluation

A. *Flow Completion Time*

For fixed-size flows (e.g., FTP, HTTP), the most attractive performance criterion is the flow completion time (FCT). Here we simulate a scenario where a large number of Poison-arriving Pareto-distributed-size flows share a single bottleneck link of 150 Mbps. The total flow number is 60000. The common Round-Trip Propagation Delay (RTPD) of all flows is 100 ms. Flows arrive as a Poison process with an average rate of 625 flows per second. The packet size is 1000 bytes. The flow sizes are Pareto distributed with a mean of 30 packets and a shape parameter of 1.2. Thus, the offered traffic load on the bottleneck link can be estimated as: 8*packet_size*mean_flow_size*flow_arrival_rate/bandwidth =1. We intentionally set the traffic load as 1 because we want to do "stress test" - high traffic load is often the situation where differentiates the performance of congestion control protocols. We record the size and completion time for each flow in the simulation, then average the flow completion time for flows with the same size.



(a) all flows



(b) flows with size < 3000 packets

Fig. 1. Average flow completion time (AFCT) vs. flow sizes for Poison-arriving Pareto-distributed-size flows. (a) is the global picture for all flows. (b) is a close look at short flows

Note that in order to make the result more accurate, we don't wait for all flows to finish, but stop the simulation just on the arrival of the 60000th flow. The reason is that the offered traffic load may drop after the arrival of the last flow, since all active flows are going to finish but no new flow is going to join. Each simulation is conducted for each protocol: TCP-Reno, XCP [6], RCP [7], and QFCP. The scenario settings and the input data (i.e., the size and arriving time of each flow) are identical for each simulation. The results show that the Average Flow Completion Time (AFCT) in QFCP is significantly shorter than that in TCP, XCP or RCP.

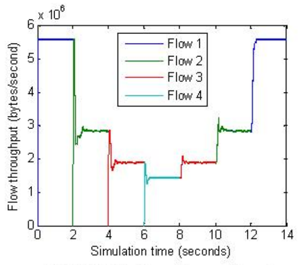
For TCP, the AFCT is very oscillatory again the flow size. The reason is that although the exponential increase of congestion window in Slow Start does help some short flows finish quickly, the duration of other flows are prolonged due to packet loss. And we should point out that Slow Start is not a good way to shorten the duration of flows, because it actually does not know the proper initial sending rate but just intends to fill up the buffer of routers and cause packet losses, which prolongs the flow duration.

For XCP, it does not use Slow Start. Instead, when new flows join, XCP tries to reclaim the bandwidth from the ongoing flows and reallocate it to the new flows little by little. For short flows they may finish before reaching the fair sending rate. That is why the completion time of short flows in XCP is the longest. However, the AFCT against flow size in XCP is more stable than in TCP because XCP flows experience fewer packet losses.

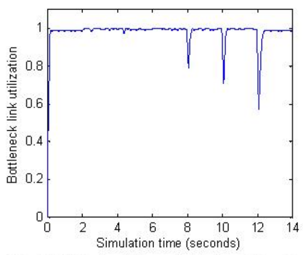
For RCP and QFCP, both of them give a high initial sending rate to new flows based on the feedback from routers and help short flows finish quickly. However, the formula used in RCP to estimate the number of flows holds only when the input traffic just fills up the link capacity *C*, otherwise it leads to wrong estimation of the flow number. This wrong estimation of flow number makes the rate allocation in RCP under-optimum and thus prolongs the FCT in general compared with QFCP.

B. Fairness and Convergence

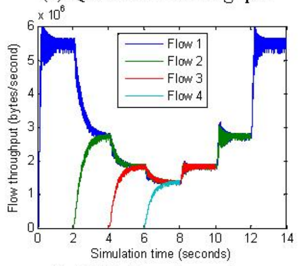
For fixed-time flows and long flows, we are more interested in the fairness of bandwidth allocation among all flows. This simulation is conducted to show the convergence and fairness of bandwidth allocation among flows, and the bottleneck link utilization for QFCP, XCP, and RCP. In this scenario, four flows share a single bottleneck link of 45 Mbps. The common RTPD of each flow is 40 ms. Flow 1-4 start at time 0, 2, 4, 6 seconds and stop at 14, 12, 10, 8 seconds respectively. The results show that QFCP can converge quickly to the fair-share sending rate as flows join and leave, and can maintain a high utilization of the bottleneck link.



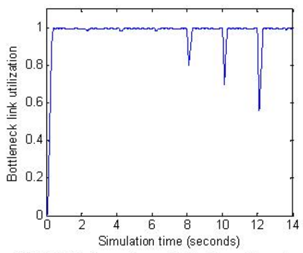
(a) QFCP: flow throughput



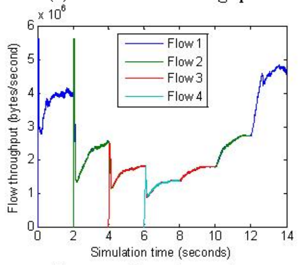
(b) QFCP: bottleneck link utilization



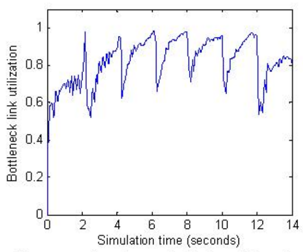
(c) XCP: flow throughput



(d) XCP: bottleneck link utilization



(e) RCP: flow throughput



(f) RCP: bottleneck link utilization

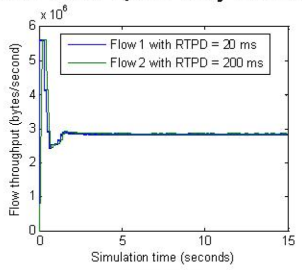
Fig. 2. Flow throughput and bottleneck link utilization

QFCP vs. RCP: The experiment shows that both protocols try to converge to the fair rate allocation if all of the flows are long enough. But due to the wrong estimation of the flow number, the utilization of bottleneck link using RCP is significantly worse than using QFCP and XCP. Moreover, RCP converges to the fair rate slower than XCP and QFCP.

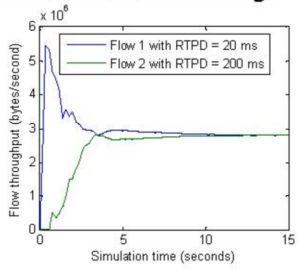
QFCP vs. XCP: The most significant difference between QFCP and XCP is the initial sending rate. In QFCP, any new flow starts with the same rate as the other ongoing flows, and then converges to the fair rate if this new flow is long enough; while in XCP, a new flow starts with a very low sending rate and then converges to the fair rate. That's why QFCP can help short flows finish within a few RTTs.

C. Flows with Variant RTTs

Previous simulation is conducted on flows with the same RTPD. Now let's see whether QFCP is robust to high variance in RTPD. There are two long-lived flows sharing a bottleneck link of 45 Mbps. The RTPD of the first flow is 20 ms, while the RTPD of the second flow is 200 ms. The result shows that although the RTPDs of the two flows are very different, QFCP only need a short time to converge to the fair rate. Compared with the result in XCP (a similar result can be found in the original XCP paper [6]), in the same scenario and using the same settings, XCP needs 10 seconds to converge to the fair rate while QFCP only needs about 1.5 seconds to converge.



(a) QFCP

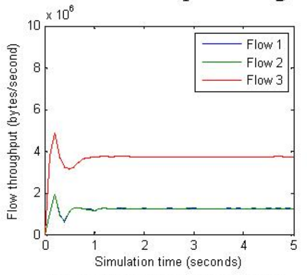


(b) XCP

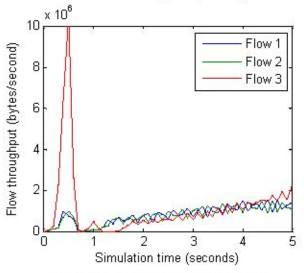
Fig. 3. Two flows with high variance in RTPD

D. *Flows Sharing Multiple Bottleneck Links*

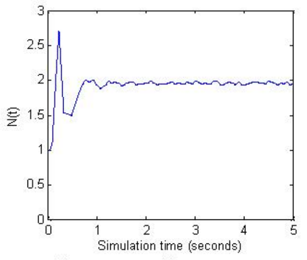
Previous simulations consider only one bottleneck link for simplicity. Now let's see whether this protocol can achieve max-min fairness when multiple bottleneck links are involved. In this simulation, there are two bottleneck links along the path. Flow1 and Flow2 share bottleneck Link1 of 20 Mbps. Flow3 is bottlenecked at Link2 of 50 Mbps. All of the three flows go through Link2. Thus, ideally the throughput of Flow1 and Flow2 should be 10 Mbps and the throughput of Flow3 should be 30 Mbps if congestion control works properly.



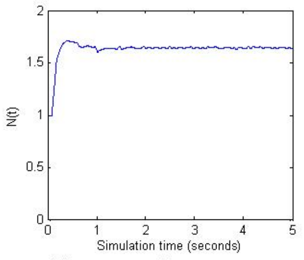
(a) QFCP: flow throughput



(b) TCP: flow throughput



(c) QFCP: N(t) on Router1



(d) QFCP: N(t) on Router2

Fig. 4. Three flows sharing two different bottleneck links

One may doubt that "Is one global rate for one output interface is enough? If some flows can not reach the sending rate assigned by a router, will the router still estimate the flow number correctly?" We should point out that the *N(t)* estimated by a router is not the actual flow number but the equivalent flow number. And *N(t)* can be any float number that greater than or equal to 1 (this minimum value 1 is set to avoid bandwidth oversubscription). Here is an example. For Router1, the estimated flow number is consistent with the real flow number 2, because both Flow1 and Flow2 can use up the rate assigned by Router1. However, for Router2, only Flow3 can send at the rate assigned by Router2, while the sending rate of the other two flows is limited somewhere else (Router1 in this case). Thus, the estimated flow number *N(t)* on Router2 is about 1.6, which is much less than the actual flow number 3. But this does not affect the stability and convergence of our protocol. Fig. 5(a) shows that all the three flows converge to their fair-share rate quickly and have maintainable high throughput. So we don't have such assumption that "all the flows must send at the assigned rate". The algorithm just wants to find out an equivalent flow number *N(t)* that best reflects the current situation of the network.

We also test the performance of TCP-Reno in this simple multiple-bottleneck-link scenario. The result shown in Fig. 5(b) confirms our previous analysis on TCP. The exponential increase of congestion window in the Slow Start phase quickly fills up the buffers of routers and causes packet drops. Then the throughput of all flows drop significantly due to packet retransmission and congestion window reduction. And flows quit the Slow Start phase and enter the congestion avoidance phase. But the AIMD algorithm grabs the available bandwidth very slowly.

IV. Conclusions

In this paper, we present the design and simulations of a new congestion control protocol QFCP. It gives a high initial sending rate for new flows as approved by routers. It can significantly shorten the completion time of both short and long flows. It can also converge to the fair-share sending rate quickly when flows join or leave. And it is robust to flows with variant RTTs. The computation overhead on routers is acceptable and most calculations only need to do periodically. Future work may include implementing QFCP in Linux and deploying it in the real networks to test its performance. And establishing some mathematical models of QFCP and doing theoretical analysis are also desirable.

References

[1] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 835-846, 1997.
[2] K. Claffy, G. Miller, and K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone," in *Proceedings of INET* '98, 1998.
[3] V. Jacobson, "Congestion avoidance and control," in Proceedings of *ACM SIGCOMM* '88, 1988.
[4] G. Liang and I. Matta, "The war between mice and elephants," in *Proceedings of ICNP* '01, 2001.
[5] S. Ebrahimi-Taghizadeh, A. Helmy, and S. Gupta, "TCP vs. TCP: a systematic study of adverse impact of short-lived TCP flows on long-lived TCP flows," in *Proceedings of INFOCOM* 2005, 2005.
[6] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM SIGCOMM* '02, 2002.
[7] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor Sharing Flows in the Internet," in *International Workshop on Quality of Service,* 2005.
[8] A. Jain, S. Floyd, M. Allman, and P. Sarolahti, "Quick-Start for TCP and IP," in *IETF Internet-draft, work in progress*, 2005.
[9] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *IEEE INFOCOM*, 2004.