APPENDIX

A Reactive Protocols may Waste Bandwidth

HPCC uses accurate in-network information, *i.e.*, queue length and link bandwidth capacity to adjust sending window. Because queue length is transient, adjustment of sending window may mismatch the current network state. We demonstrate how accurate information may lead to inaccurate bandwidth allocation by conducting simulations where small flows arrive periodically. There are two Hosts A and B transmitting messages to one Host C. Host A generates small flows whose size is one BDP (*i.e.*, 7 MTU) periodically, while Host B transmits a large flow. Bandwidth utilization on Host C is shown in Figure 14. Two lines correspond to different flow's arriving intervals on Host A. When one small flow and one long flow compete in the network bottleneck, the queue builds up. INT carries back network state to Host A and Host B, and the sending window of the long flow decreases. Unfortunately, the small flow on Host A has already finished and does not use bandwidth anymore. Therefore, the bandwidth on Host C can be wasted. Waste can stack up when small flows arrive at the same time, as the orange line shows.

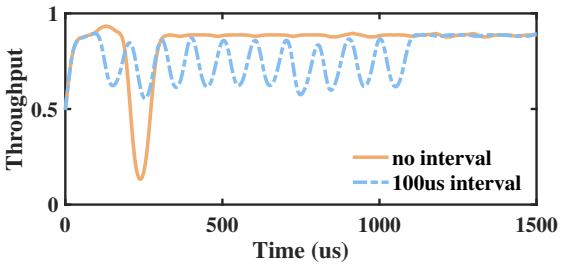


Figure 14. Reactive protocols may waste bandwidth.

B An Example to Illustrate the Debt Mechanism

In § 4.1, we discussed the debt mechanism. Figure 15 illustrates the debt mechanism by leveraging a simple example. The upper black arrows stand for generations. Assume the BDP is two MTU packets, then u/s denotes unscheduled/scheduled packets, and d/t indicates debts/tokens, respectively. u_0, u_1, and s_0 are flagged with `token_request`. The last BDP worth of data (*i.e.*, s_1 and s_2) is not flagged. When u_0 is received, the receiver generates one debt d_0 and one token t_0 (due to the flag). It is the same case for u_1. When s_0 is received, one token t_2 is generated by the receiver. When s_1/s_2 is received, no token is generated. With this design, the flow size is not required to be known a-prior.

Figure 15 depicts the triggering logic of debts/tokens, where $u0$ and $s1$, which have red borders, are ECN-marked. Red arrows stand for triggering action, *i.e.*, each ECN-marked data packet triggers the transmission of a debt that is already generated by unscheduled packets. Receiver paces debts/tokens at 5% of port bandwidth (§4.1). If the debt is triggered, send it first. Otherwise, send a token instead.

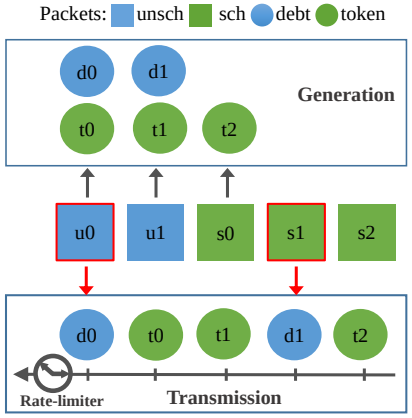
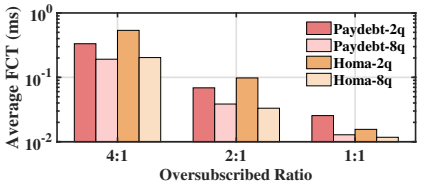
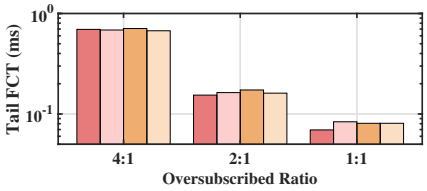


Figure 15. Debt mechanism.



(a) Average FCT.



(b) Tail latency.

Figure 16. Performance of Memcached under different topologies. SRPT scheduling and 8 priority queues are leveraged.

C Leveraging Prioritization to PayDebt

Homa leverages SRPT scheduling and in-network priority queues to achieve good performance on small flows. To dig into the performance of PayDebt under prioritization, we configure PayDebt with the same SRPT scheduling and eight priority queues as used in Homa. Figure 16 shows the performance of PayDebt and Homa under Memcached across different fan-in ratios. By adding SRPT strategy and in-network priority queues, PayDebt (8q) achieves a better performance on average FCT compared with PayDebt (2q). PayDebt (8q) achieves comparable performance compared to Homa (8q). By increasing the number of priority queues, the benefit of achieving a smaller buffer occupancy is mitigated, *i.e.*, the difference between Homa and PayDebt shrinks when 8 priority queues are used. This is expected because a larger number of priority queues could reduce the queuing delay caused by buffer built-up.