Forty-Sixth Annual Allerton Conference
Allerton House, UIUC, Illinois, USA
September 23-26, 2008

Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization

Mohammad Alizadeh, Berk Atikoglu, Abdul Kabbani, Ashvin Lakshmikantha, Rong Pan
Balaji Prabhakar, and Mick Seaman

Abstract— **Data Center Networks present a novel, unique and rich environment for algorithm development and deployment. Projects are underway in the IEEE 802.1 standards body, especially in the Data Center Bridging Task Group, to define new switched Ethernet functions for data center use.**

One such project is IEEE 802.1Qau, the Congestion Notification project, whose aim is to develop an Ethernet congestion control algorithm for hardware implementation. A major contribution of this paper is the description and analysis of the congestion control algorithm—QCN, for Quantized Congestion Notification— which has been developed for this purpose.

A second contribution of the paper is an articulation of the Averaging Principle: a simple method for making congestion control loops stable in the face of increasing lags. This contrasts with two well-known methods of stabilizing control loops as lags increase; namely, (i) increasing the order of the system by sensing and feeding back higher-order derivatives of the state, and (ii) determining the lag and then choosing appropriate loop gains. Both methods have been applied in the congestion control literature to obtain stable algorithms for high bandwidth-delay product paths in the Internet. However, these methods are either undesirable or infeasible in the Ethernet context. The Averaging Principle provides a simple alternative, one which we are able to theoretically characterize.

I. Introduction

Data centers have emerged in the past few years as a new paradigm for interconnecting computing and storage on a massive scale. There are several viewpoints from which to approach the development of data centers: as the outgrowth of large web server farms (for web-hosting), as the convergence of computing and networking (high-performance computing as typified by the Cloud Computing paradigm), and as a convergence of local area networks and storage networks. Several technological innovations have spurred the rapid deployment of data centers; notably, 10 Gbps Ethernet technology, the specification of Fiber Channel over Ethernet (FCoE) standards, server virtualization, and the development of high-performance Network Interface Cards (NICs).

Mohammad Alizadeh and Abdul Kabbani are with the Department of Electrical Engineering, Stanford University {alizade, akabbani}@stanford.edu

Berk Atikoglu is with the Department of Computer Science, Stanford University atikoglu@stanford.edu

Ashvin Lakshmikantha is with Broadcom Corporation, San Jose, California ashvinla@broadcom.com

Rong Pan is with Cisco Systems, San Jose, California ropan@cisco.com

Balaji Prabhakar is with the Departments of Electrical Engineering and Computer Science, Stanford University balaji@stanford.edu

Mick Seaman is the Chair, Security Task Group; Chair, Interworking Group (1986–2007), IEEE 802.1 mickseaman@SBCGLOBAL.NET

The Data Center Bridging Task Group in the IEEE 802.1 Ethernet standards body has several active projects aimed at enabling enhancements to classical switched Ethernet so that it may provide more services. Details of the DCB Task Group's projects are available at [17]. Particularly relevant to the present paper are the Congestion Notification (CN, IEEE 802.1Qau) and the Priority-based Flow Control (PFC, IEEE 802.1Qbb) projects. The first project is concerned with the specification of a Layer 2 congestion control mechanism, in which a congested switch can control the rates of Layer 2 sources whose packets are passing through the switch. Thus, the CN project induces congestion control loops at Layer 2 similar to the well-known TCP/RED control loops at Layer 3. The second project is concerned with introducing a link-level, per-priority flow control or PAUSE function.

Whereas the CN and PFC projects are functionally similar to previous work, the operating conditions in switched Ethernets vastly differ in ways that will be made clear in the next section. This has necessitated the development of a novel congestion control scheme, called QCN (for Quantized Congestion Notification). A primary goal and contribution of this paper is to describe the QCN algorithm and to present a mathematical model useful for understanding its stability. The QCN algorithm shares commonalities with the BIC-TCP algorithm [15] at the source and the REM [4] and PI [5] controllers at the switch; therefore, the analytical model we develop for QCN is useful for understanding BIC-TCP as well.

A second contribution of this paper is the articulation of the Averaging Principle, which is a simple method for improving the stability of congestion control loops in the presence of increasing lags (or round trip times, RTTs) such as can occur in high bandwidth-delay product networks. It is well-known that the stability of a control loop worsens as the lag between the source and the network increases. Two main methods of feedback compensation are employed for restoring stability at large RTTs: (i) determine the RTT and choose appropriate control gain parameters, and (ii) enrich the state of the system by feeding back (linear combinations of) higher order derivatives of the queue-size process at network switches and routers. Most existing congestion control algorithms for high bandwidth-delay product networks can be classified as being of one or the other type. See Section III for details.

The Averaging Principle is another method, which neither requires knowledge of the RTT nor needs the network to be upgraded. We apply the Averaging Principle to an

978-1-4244-2926-4/08/$25.00 ©2008 IEEE

AIMD (for additive increase, multiplicative decrease) scheme and see that it dramatically improves the stability of the AIMD scheme. We demonstrate that the Averaging Principle is equivalent to a scheme which feeds back higher order derivatives of the queue-size process. This initial result hints at the fundamental reason for the good performance of the Averaging Principle and encourages a deeper exploration.

Of necessity, this paper is brief. We have presented the main points of the QCN algorithm and the Averaging Principle deferring detailed treatments to further publications. We are particularly interested in exploring the Averaging Principle in greater detail and in seeking out applications for it outside the congestion control context.

We conclude the introduction by noting that Data Center Networks provide an excellent opportunity for revisiting some of the basic issues of packet switched networks, such as congestion control, switching, forwarding/routing, measurement and traffic engineering. They have the scale, in terms of the number of nodes, of a large subnetwork of the Internet, they operate under new and unique constraints, and they aim to support novel services, applications and technologies. This makes Data Center Networks a really interesting research subject.

II. QCN

The QCN (Quantized Congestion Notification) algorithm has been developed to provide congestion control at the Ethernet layer, or at L2. It has been developed for the IEEE 802.1Qau standard, which is a part of the IEEE Data Center Bridging Task Group's efforts. A related effort is the Priority Flow Control project, IEEE 802.1Qbb, for enabling hop-by-hop, per-priority pausing of traffic at congested links. Thus, when the buffer at a congested link fills up, it issues a PAUSE message to upstream buffers, an action which ensures packets do not get dropped due to congestion. A consequence of link-level pausing is the phenomenon of "congestion spreading:" the domino effect of buffer congestion propagating upstream causing secondary bottlenecks. Secondary bottlenecks are highly undesirable as they affect sources whose packets do not pass through the primary bottleneck. An L2 congestion control scheme allows a primary bottleneck to directly reduce the rates of those sources whose packets pass through it, thereby preventing (or reducing the instances of) secondary bottlenecks. The L2 congestion control algorithm is expected to operate well regardless of whether link-level pause exists or not (i.e. packets may be dropped).

Many differences exist between the operating environments of the Internet and switched Ethernet, which we list below. These differences and performance requirements place some unique restrictions on the type of L2 congestion control that is suitable for the Data Center environment.

Switched Ethernet vs. the Internet. There are several differences, we only list the most important below. One major issue which we will only mention briefly is multipathing, which is a key feature in Ethernet and which quite severely affects both the design and the performance of congestion control schemes.

1. *No per-packet acks in Ethernet.* This has several consequences for congestion control mechanisms: (i) Packet transmission is not self-clocked as in the Internet, (ii) path delays (round trip times) are not knowable, and (iii) congestion must be signaled by switches directly to sources. The last point makes it difficult to know *path* congestion; one only knows about *node* congestion.
2. *Packets may not be dropped.* As mentioned, Ethernet links may be paused and packets may not be dropped. A significant side-effect of this is that congestion spreading can occur, causing spurious secondary bottlenecks.
3. *No packet sequence numbers.* L2 packets do not have sequence numbers from which RTTs, or the length of the "control loop" in terms of number of packets in flight, may be inferred.
4. *Sources start at the line rate.* Unlike the slow-start mechanism in TCP, L2 sources may start transmission at the full line rate of 10 Gbps. This is because L2 sources are implemented in hardware, and installing rate limiters is the only way to have a source send at *less* than the line rate. But since rate limiters are typically few in number, it is preferable to install them only when a source gets a congestion message from a switch.
5. *Very shallow buffers.* Ethernet switch buffers are typically 100s of KBytes deep, as opposed to Internet router buffers which are 100s of MBytes deep. Even though in terms of bandwidth-delay product the difference is about right (Ethernet RTTs are a few 100 μsecs, as opposed Internet RTTs which are a few 100 msecs), the transfer of a single file of, say, 1 MByte length can overwhelm an Ethernet buffer. This is especially true when L2 sources come on at the line rate.
6. *Small number-of-sources regime is typical.* In the Internet literature on congestion control, one usually studies the system when the number of sources is large, which is typical in the Internet. However, in Ethernet (especially in Data Centers), it is the small number of sources that is typical. This imposes serious constraints on the stability of congestion control loops, see below.
7. *Multipathing.* Forwarding in Ethernet is done on spanning trees. While this avoids loops, it is both fragile (there is only one path on a tree between any pair of nodes) and leads to an underutilization of network capacity. For these reasons, equal cost multipathing (ECMP) is some times implemented in Ethernet. In this scenario there is more than one path for packets to go from an L2 source to an L2 destination. However, congestion levels on the different paths may be vastly different!

Performance requirements. The congestion control algorithm should be

a. *Stable.* This means buffer occupancy processes should not fluctuate, causing overflows and underflows. Such episodes either lead to dropped packets or to link underutilization. This is particularly important when trying to control a small number of high bandwidth

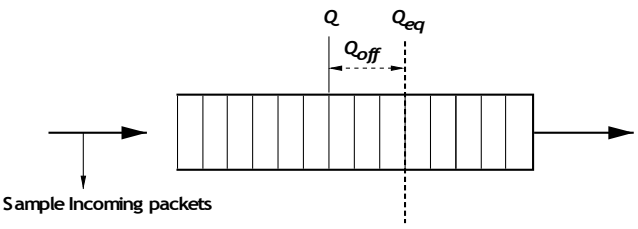

Fig. 1: Congestion detection in QCN CP.

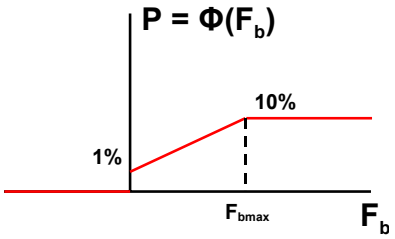

Fig. 2: Sampling (reflection) probability in QCN CP as a function of $|F_b|$.

sources with a shallow buffer, whose depth is a fraction of the bandwidth-delay product. For example, we would like to operate switch buffers at 30 KByte occupancy when a single 10 Gbps source is traversing it and the overall RTT is 500 μsecs. That is, we aim to keep the buffer occupancy at less than 6% of the bandwidth-delay product!

b. *Responsive.* Ethernet link bandwidth on a priority can vary with time due to traffic fluctuation in other priorities, the appearance of bottlenecks due to pause, the arrival of new sources, etc. These variations can be extreme: from 10 Gbps to 0.5 Gbps and back up again. The algorithm needs to rapidly adapt source rates to these variations.

c. *Fair.* When multiple flows share a link, they should obtain nearly the same share of the link's bandwidth.

d. *Simple to implement.* The algorithm will be implemented entirely in hardware. Therefore, it should be very simple. A corollary of this requirement is that complicated calculations of rates, control loop gains and other "variables" should be avoided.

A. The QCN Algorithm

We shall now describe the QCN algorithm. We focus on its key features and omit a number of details important for an exact implementation. Those interested are referred to the QCN pseudo-code [19].

The algorithm is composed of two parts:

(i) Switch or Congestion Point (CP) Dynamics: this is the mechanism by which a switch buffer attached to an oversubscribed link samples incoming packets and generates a feedback message addressed to the source of the sampled packet. The feedback message contains information about the extent of congestion at the CP.

(ii) Rate limiter or Reaction Point (RP) Dynamics: this is the mechanism by which a rate limiter (RL) associated with a source decreases its sending rate based on feedback received from the CP, and increases its rate *voluntarily* to recover lost bandwidth and probe for extra available bandwidth.

The CP Algorithm

Following the practice in IEEE standards, we think of the CP as an ideal output-buffered switch even though actual implementations may differ. The CP buffer is shown in Fig. 1. The goal of the CP is to maintain the buffer occupancy at a desired operating point, Q_{eq}.[1] The CP computes a congestion measure F_b (defined below) and, with a probability depending on the severity of congestion, randomly samples[2] an incoming packet and sends the value of F_b in a feedback message to the source of the sampled packet. The value of F_b is quantized to 6 bits.

Let Q denote the instantaneous queue-size and Q_{old} denote the queue-size when the the last feedback message was generated. Let $Q_{off} = Q - Q_{eq}$ and $Q_\delta = Q - Q_{old}$. Then F_b is given by the formula

$$F_b = -(Q_{off} + wQ_\delta), \tag{1}$$

where w is a non-negative constant, taken to be 2 for the baseline implementation.

The interpretation is that F_b captures a combination of queue-size excess (Q_{off}) and rate excess (Q_δ). Indeed, $Q_\delta = Q - Q_{old}$ is the *derivative* of the queue-size and equals input rate less output rate. Thus, when F_b is negative, either the buffers or the link or both are oversubscribed. When $F_b < 0$, Fig. 2 shows the probability with which a congestion message is reflected back to the source as a function of $|F_b|$. The feedback message contains the value of F_b, quantized to 6 bits. When $F_b \geq 0$, there is no congestion and no feedback messages are sent.

The RP Algorithm

Since the RP is not given positive rate-increase signals by the network, it needs a mechanism for increasing its sending rate on its own. Due to the absence of acks in Ethernet, the increases of rate need to be clocked internally at the RP. Before proceeding to explain the RP algorithm, we will need the following terminology:

- *Current Rate (CR)*: The transmission rate of the RL at any time.
- *Target Rate (TR)*: The sending rate of the RL *just before* the arrival of the last feedback message.
- *Byte Counter*: A counter at the RP for counting the number of bytes transmitted by the RL. It times rate increases by the RL. See below.
- *Timer*: A clock at the RP which is also used for timing rate increases at the RL. The main purpose of the timer is to allow the RL to rapidly increase when its sending

[1] It is helpful to think of Q_{eq} as roughly equal to 20% of the size of the physical buffer.

[2] The actual implementation is slightly different; refer to [19] for details.

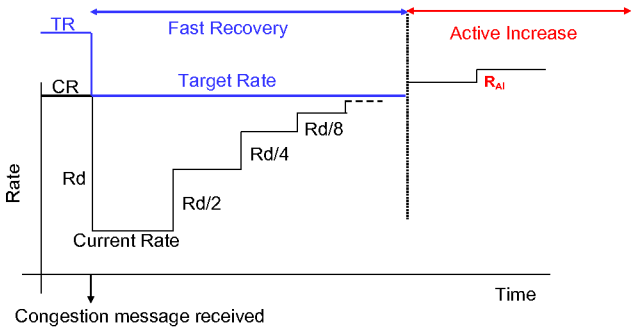

Fig. 3: QCN RP operation.

rate is very low and there is a lot of bandwidth becomes available. See below.

We now explain the RP algorithm assuming that only the byte counter is available. Later, we will briefly explain how the timer is integrated into the RP algorithm. Fig. 3 shows the basic RP behavior.

Rate decreases. This occurs only when a feedback message is received, in which case CR and TR are updated as follows:

$$CR \leftarrow CR(1 - G_d|F_b|) \tag{2}$$

$$TR \leftarrow CR, \tag{3}$$

where the constant G_d is chosen so that $G_d|F_{bmax}| = \frac{1}{2}$, i.e. the sending rate can decrease by at most 50 %.

Rate increases. This occurs in two phases: Fast Recovery and Active Increase.

Fast Recovery (FR). The byte counter is reset every time a rate decrease is applied and it enters the FR state. FR consists of 5 cycles, each cycle equal to 150 KBytes of data transmission by the RL. The cycles are counted by the byte counter. At the end of each cycle, TR remains unchanged while CR is updated as follows:

$$CR \leftarrow \frac{1}{2}(CR + TR). \tag{4}$$

The cycle duration of 150 KBytes is chosen to correspond to the transmission of 100 packets, each 1500 Bytes long. The idea is that when the RL has transmitted 100 packets and, given that the minimum sampling probability at the CP is 1%, if it hasn't received a feedback message then it may infer that the CP is uncongested. Therefore it increases its rate as above, recovering some of the bandwidth it lost at the previous rate decrease episode. Thus, the goal of the RP in FR is to *rapidly recover* the rate it lost at the last rate decrease episode.

Note. The BIC-TCP algorithm is the first to introduce Fast Recovery. We have independently, but subsequently, rediscovered it.

Active Increase (AI). After 5 cycles of FR have completed, the RP enters the Active Increase (AI) phase where it probes for extra bandwidth on the path. During AI, the byte counter counts out cycles of 50 packets (this can be set to 100 packets for a less frequent probing). At the end of a feedback message, the RL updates TR and CR as follows:

$$TR \leftarrow TR + R_{AI} \tag{5}$$

$$CR \leftarrow \frac{1}{2}(CR + TR), \tag{6}$$

where R_{AI} is a constant chosen to be 5 Mbps in the baseline implementation. This completes the description of the basic RP algorithm using only the Byte Counter.

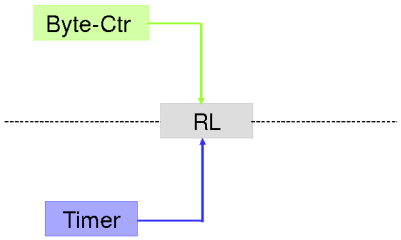

Fig. 4: Byte Counter and Timer interaction with RL.

Supplementary Notes and Remarks

Remark 1. Since rate increases using the Byte Counter occur at times proportional to the current sending rate of the RL, when the CR is very small, the duration of Byte Counter cycles when measured in seconds can become unacceptably large. Since the speed of bandwidth recovery (or responsiveness) is a key performance metric, we have included the Timer in QCN.

The Timer functions similarly as the Byte Counter: it is reset when a feedback message arrives, enters FR and counts out 5 cycles of T ms duration (T is 10 ms long in the baseline), and then enter AI where each cycle is $T/2$ ms long.

The Byte Counter and Timer jointly determine rate increases at the RL as shown in Fig. 4. After a feedback message is received, they each operate independently and execute their respective cycles of FR and AI. Together, they determine the state of the RL and its rate changes as follows.

1. The RL is in FR if *both* the Byte Counter and the Timer are in FR. In this case, when either the Byte Counter or the Timer completes a cycle, CR is updated according to (4).
2. The RL is in AI if *only one of* the Byte Counter and the Timer is in AI. In this case, when either completes a cycle, TR and CR are updated according (5) and (6).
3. The RL is in HAI (for Hyper-Active Increase) if *both* the Byte Counter and the timer are in AI. In this case, the ith time that a cycle for either the Byte Counter or the Timer is completed, TR and CR are updated as:

$$TR \leftarrow TR + iR_{HAI} \tag{7}$$

$$CR \leftarrow \frac{1}{2}(CR + TR) \tag{8}$$

where R_{HAI} is constant set to 50 Mbps in the baseline implementation. So the increments to TR (and hence of CR) in HAI occur in multiples of 50 Mbps.

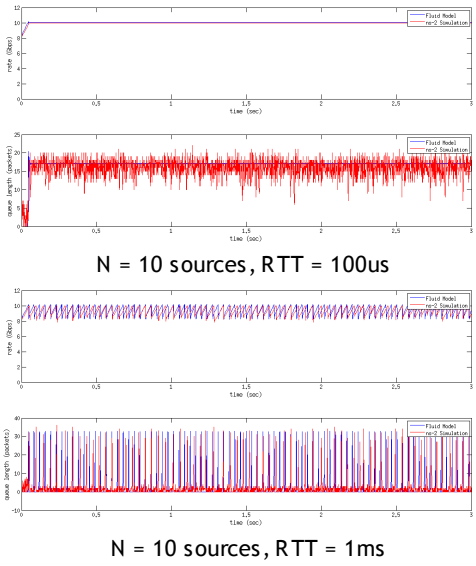

Fig. 5: Comparison of QCN fluid model and ns-2 simulation

Thus, the Byte Counter and Timer should be viewed as providing "rate increase instances" to the RL. Their state determines the state and, hence, the amount of rate increase at the RL.

Remark 2. It is very important to note that the RL goes to HAI only after at least 500 packets have been sent *and* 50 msecs have passed since the last congestion feedback message was received. This doubly ensures that aggressive rate increases occur only after the RL provides the network adequate opportunity (in packets sent for possible sampling) for sending rate decrease signals should there be congestion. This is vital to ensure the stability of the algorithm, and while optimizations can be performed to improve its responsiveness, in the interests of stability and simplicity, we have resisted the temptation to optimize.

Remark 3. Other crucial features of the QCN algorithm (such as Extra Fast Recovery and Target Rate Reduction) which are useful for ensuring its reliable performance when the path bandwidth *suddenly drops* have been omitted here.

Remark 4. The manner of rate increases in QCN during FR is the central innovation of the BIC-TCP algorithm. Therefore, the model for QCN's evolution, presented next, can be twinned with that of BIC. This would be the first time that the familiar delay-differential equation model of TCP can be written down for BIC. The conceptual point needed for obtaining the delay-differential equation model was the recognition that there are two *independent* variables describing source dynamics: TR and CR. There is usually only one variable at the source: its current sending rate (or window size). Since BIC-TCP has an explicit notion of time, in the form of rate increases occurring once every RTT, a Timer-only version of QCN would be its counterpart.

B. QCN Fluid Model

The fluid model derived below corresponds with a simplified version of QCN, with some features pertaining to the use of the Timer and its transient evolution disabled. Due to space constraints we also do not explain the derivation of the equations; for the most part, this is part of the research literature. The main difference is in our use of two variables, TR and CR, to represent source behavior. As mentioned earlier, this is a necessary step, since both TR and CR are *independent* variables, although they are inter-dependent.

We compare the accuracy of the model with a packet-level simulation of QCN using the ns-2 simulator [21]. Consider a "dumbbell topology" with N sources sharing a single link. Source i's TR and CR are denoted by TR_i and CR_i, and their evolution is given by the following differential equations:

$$\begin{aligned}\frac{dTR_i}{dt} =& -(TR_i(t)-CR_i(t))CR_i(t-\tau)p(t-\tau)\\ &+(1-p(t-\tau))^{500}R_{AI}\frac{CR_i(t-\tau)}{100} \qquad (9)\\ \frac{dCR_i}{dt} =& -(G_dF_b(t-\tau)CR_i(t))CR_i(t-\tau)p(t-\tau)\\ &+(\frac{TR_i(t-\tau)-CR_i(t-\tau)}{2})\frac{CR_i(t-\tau)p(t-\tau)}{\frac{1}{(1-p(t-\tau))^{100}}-1} \qquad (10)\end{aligned}$$

where $p(t)$ is the time-varying sampling probability at the switch, and $F_b(t)$ is the congestion price. These quantities evolve according the the switch dynamics given by:

$$\frac{dq}{dt} = \sum_{i=1}^{N} CR_i(t) - C \qquad (11)$$

$$F_b(t) = q(t) - Q_{eq} + \frac{w}{Cp(t)}(\sum_{i=1}^{N} CR_i(t) - C) \qquad (12)$$

$$\frac{dp}{dt} = 500(\Phi(F_b(t)) - p(t)) \qquad (13)$$

where $\Phi()$ is the function shown in Fig. 2.

Fig. 5 compares the QCN fluid model with ns-2 simulations for the same scenarios. As can be seen, there is a very good match between the model and simulations.

III. Congestion Control in High Bandwidth-Delay Product Networks

A. Background

The difficulties of designing efficient and stable congestion control protocols for high bandwidth-delay product networks have been well-documented in the literature. In particular it is well-known that, with long-lived flows, as lags in the congestion control loop increase, network queues may become oscillatory and prone to instability. See, for example, [12] and [13]. This has spawned many new congestion control algorithms: [11], [14], [1], [2], [3], [15] and [16].

These problems are further magnified in the data center environment due to the following two important constraints:

(i) the network is operating with shallow buffers, and (ii) there are typically a small number of sources active on each path. While the first constraint places a rigid limit on the magnitude of oscillations, the second implies the active sources are of very high rate, ranging from 10% to 100% of the line rate, making it difficult to tightly control their sending rates. This makes designing transport mechanisms for data centers an interesting challenge, requiring the development of new algorithms and congestion control theory.

Let us now consider a general control system (i.e., not necessarily a *congestion* control system) operating in the presence of increasing lags between the controller and the plant. It is well-known that such a system loses stability margin as lags increase and feedback compensation needs to be applied to restore stability. There are two major flavors of feedback compensation: (i) knowing the lags, the system can adjust the loop gains so as to restore stability, and (ii) the state-space is enriched at the plant with the addition of higher order derivatives of the original state. Roughly speaking, the former leaves the state unchanged while the latter leaves the gains unchanged.

When particularized to congestion control loops with large lags (or, equivalently, with high bandwidth-delay products), feedback compensation has spawned two main approaches. In one approach, an estimate of the round trip time is used to find the correct "gains" for the loop to be stable; for example, this is the approach taken by XCP [1], RCP [2], and FAST [3]. The second approach aims to improve the loop stability by including more queue information in the congestion prices. For instance, the active queue management protocols REM [4], and PI [5] both use a weighted sum of the queue size and the net input rate to the queue (or the queue derivative) as the congestion price. This approach has also been adopted by the BCN [9] protocol to improve its stability relative to an earlier version of the same algorithm where just queue information was fed back. The first approach is not feasible in Ethernet networks since there are no per-packet acknowledgments from which round trip times may be inferred. The second approach introduces a non-trivial change at the switches and routers, which makes it undesirable.

This discussion leads to the following interesting question: Is there a source-side algorithm which does not change loop gains (since round trip times are not available) and which improves the control loop stability?

We are encouraged to seek an answer in the affirmative because of the BIC-TCP and QCN algorithms. BIC-TCP operates stably in high-bandwidth delay product networks, even though it changes neither loop gains nor router behavior. However, it does operate in the self-clocked universe of Internet congestion control schemes, where window size changes are made once every round trip time. So there is the possibility that it implicitly exploits a knowledge of round trip times to derive stability. The QCN algorithm has no notion of round trip times! The similarity of operation of the BIC-TCP and QCN algorithms suggests they are stable for large lags for a more fundamental reason. Our attempt to uncover and understand this reason has led us to the "Averaging Principle," which we articulate in the next section.

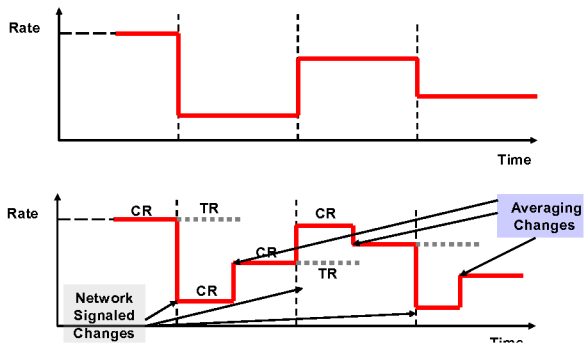

Fig. 6: Basic (top) and AP enabled source (bottom)

B. The Averaging Principle

We model a congestion control loop as a sampled-data control system; i.e., a system in which the state or the plant output is sampled periodically and fed back to the controller. Such a model is appropriate for congestion control schemes where packets are sampled periodically (albeit randomly) at the switches/routers and feedback is sent to the corresponding sources. We assume that the sampling period is known to the sources; for example, we take it as the reciprocal of the minimum sampling probability at the switches, which is a well-known quantity. Denote by T the length of the sampling period counted in number of transmitted packets. For emphasis, we note that T is a *deterministic* quantity, even though sampling may occur randomly with a mean sampling period larger than equal to T.

It is to be noted that Internet congestion control schemes typically change sending rates in multiples of *round trip times (RTTs)*, and not in multiples of packets sent. Therefore, it is natural in such networks for T to have units of time (or RTTs). However, in keeping with the Ethernet-centric nature of this paper, we shall take T to be counted in packets transmitted.

We shall use the notions of Current Rate (CR) and Target Rate (TR) introduced earlier, see Fig. 6 for an illustration.

The Averaging Principle (AP), Basic form.
In addition to changing the CR upon the receipt of feedback messages from the network, the AP stipulates that CR be changed as follows: After $T/2$ packets have been transmitted since the receipt of the last feedback message, change CR to $(CR+TR)/2$. In other words, halfway during the sampling interval, let CR move halfway towards TR (which was the value of the CR before the receipt of the last feedback message).

Remark 1: The statement of the basic form of the AP works best when the congestion control loop corresponds naturally with a sampled-data system. Thus, it works best when signals arrive from the network regularly, about once every T transmitted packets. Schemes like BCN, RCP and

XCP in which the network sends both rate increase and decrease signals at regular intervals are, therefore, naturally amenable to an application of the AP. Other schemes, such as TCP, where the network only sends rate decrease signals can have a more variable period between feedback messages. Here too the AP applies (witness BIC-TCP and QCN), albeit in a slightly different form. We defer an elaboration of this point to further publications.

Remark 2: One could set CR equal to $\alpha CR + (1-\alpha)TR$, where $\alpha \in (0,1)$ and apply averaging at points other than the midpoint of the sampling interval. Other generalizations are possible.

Effectiveness of the AP: A case study.

To demonstrate the effectiveness of the AP, we will apply it to an AIMD scheme in which rate increases and decreases are signaled by the network. This scheme is a version of the BCN algorithm[3] studied by the IEEE DCTG, with some minor features disabled. We shall refer to it as N-AIMD, for Network-AIMD. The AP applies equally well to N-MIMD schemes; although, due to a shortage of space we do not present those results here.

In N-AIMD, switches randomly sample packets with 1% probability. Thus, $T = 100$ packets and the AP is applied 50 packets after the last feedback message was received by the source. We use a packet counter at the source which is reset every time a feedback message is received, and which is incremented for every transmitted packet. Averaging is applied when the count reaches 50, after which the CR is unchanged until the next feedback message is received. If the source receives a feedback message before the packet counter has reached 50, the counter is reset and no averaging takes place.

When a packet is sampled at the switch, an F_b value is computed in exactly the same way as in equation (1). However, whether F_b is positive or negative, N-AIMD sends the F_b value to the source of the sampled packet. Upon receipt of the feedback message, the source updates its rate as:

$$R \leftarrow \begin{cases} R + G_i R_u F_b & \text{if } F_b \geq 0 \quad (14a) \\ R(1 - G_d|F_b|) & \text{if } F_b < 0 \quad (14b) \end{cases}$$

where G_i, R_u, and G_d are non-negative parameters of the protocol.

Averaging Principle Simulation.

We compare the N-AIMD scheme to the scheme which results from applying the AP to N-AIMD (AP-N-AIMD). We use the packet-level simulator OMNeT++ [10], and the parameter values of the baseline BCN implementation [18], namely: $Q_{eq} = 16$ packets, $w = 2$, $G_i = 0.53333$, $R_u = 1000000$, $G_d = 0.0026667$. We adopt the conventional dumbbell topology with a single bottleneck link and compare the two schemes under increasing RTTs.

First consider the small number-of-sources-regime. Fig. 7 shows the instantaneous queue sizes for the case where

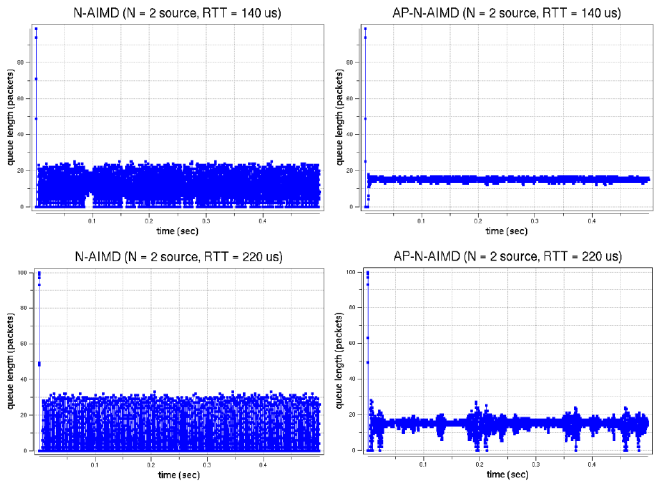

Fig. 7: N = 2 sources: N-AIMD (left) vs. AP-N-AIMD (right)

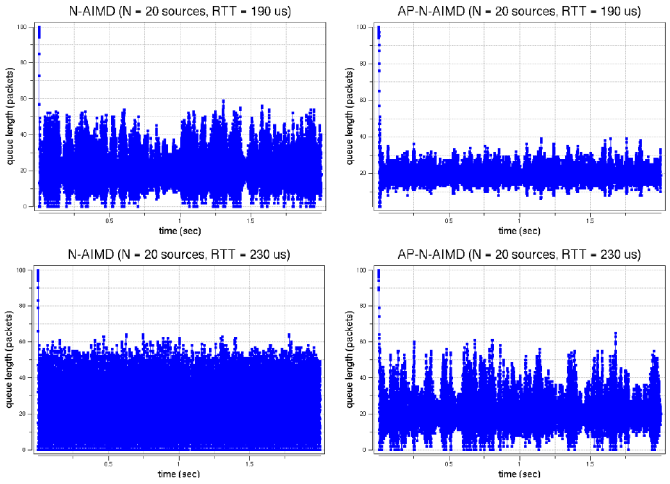

Fig. 8: N = 20 sources: N-AIMD (left) vs. AP-N-AIMD (right)

2 sources share a bottleneck link with 10 Gbps capacity. As can be seen, when the RTT is increased to 140 μs, N-AIMD is no longer stable, while AP-N-AIMD remains quite stable, keeping the queue oscillations small around $Q_{eq} = 16$ packets. It isn't until an RTT of 240 μs that the AP-N-AIMD also loses its stability, and at this RTT N-AIMD is unstable, with underflowing queues leading to substantial link under-utilization.

Now consider a simulation with 20 sources[4] as depicted in Fig. 8. In this case, N-AIMD becomes unstable at an RTT of 190 μs. But AP-N-AIMD becomes unstable only when the RTT is increased beyond 230 μs. Hence in this case as well, the AP improves protocol stability. As the number of sources increases, the margin of improvement diminishes because the link is quite well utilized by the aggregate of sources even under N-AIMD, leaving little margin for AP to improve.

[3]For a control-theoretic analysis of BCN, see [9].

[4]20 is a large enough number of sources in the Ethernet context.

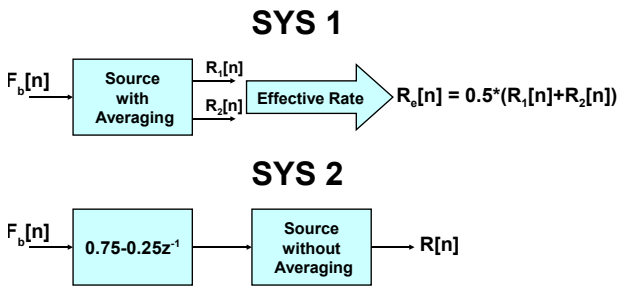

Fig. 9: Single source Averaging Principle (top) and its equivalent (bottom) systems

C. Analysis of the Averaging Principle

So, why does the AP work well in providing feedback compensation? We now provide an initial answer to this question. The punch-line is: the AP is equivalent to a scheme which adds an extra derivative of the queue-size (the second derivative in the case of N-AIMD) to the state. The latter is a well-known plant- or switch-side method of increasing the stability margin; however, it requires a non-trivial change to the way switches operate. Due to lack of space, we will not go into the details and proof of this result, but rather just state a basic representative theorem for the single source case.

Consider the two systems shown in Fig. 9. In System 1, the source receives the feedback sequence $F_b[n]$ and adjusts its sending rate according to the AP. $R_1[n]$ and $R_2[n]$ are the source's sending rate before and after the averaging change is applied and after receiving the nth feedback sample, $F_b[n]$. $R_e[n]$ is the mean of $R_1[n]$ and $R_2[n]$ which is clearly the effective rate the switch queue sees, since it acts as an integrator. In System 2, the source does not employ the AP, but uses the value $\frac{3}{4}F_b[n] - \frac{1}{4}F_b[n-1]$ to determine its rate adjustments.

Theorem 1: Systems 1 and System 2 are algebraically equivalent; that is, if the two systems have the same initial rate ($R_e[0] = R[0]$) and are given the same feedback sequence $(F_b[n]; n \in \mathbb{N})$, they produce identical sequences of sending rates, i.e. $R_e[n] = R[n]$ for all $n \geq 1$.

Remark: Since

$$\begin{aligned}\frac{3F_b[n] - F_b[n-1]}{4} &= \frac{F_b[n]}{2} + \frac{F_b[n] - F_b[n-1]}{4} \\ &\approx \frac{F_b}{2} + \frac{T}{4}\frac{dF_b}{dt} \end{aligned} \quad (15)$$

where T is the duration of a single sample interval.

Theorem 1 shows the AP results in a more stable system, because it is equivalent to a system which feeds back the *second* derivative of the queue-size (i.e. $\frac{dF_b}{dt} = \frac{dQ}{dt} + w\frac{d^2Q}{dt^2}$).

IV. Conclusion

Data Center Networks are exciting to the industry and for research. It affords the opportunity for the development and deployment of new networking ideas. One development described in this paper is the QCN L2 congestion control algorithm. The analytical model of QCN is novel in that it includes a "memory" element at the source and it can be used to study BIC-TCP. In trying to understand the fundamental reason for the good stability of QCN and BIC, we were led to uncover the Averaging Principle and obtain a theoretical understanding of the same.

V. Acknowledgment

Mohammad Alizadeh is supported by the Numerical Technologies, Inc. Fellowship and a Stanford Graduate Fellowship. We thank Sanaz Motahari for simulations related to the Averaging Principle, and members of the IEEE DCTG for various discussions of QCN; notably, Davide Bergamasco, Mitch Gusat, Bruce Kwan, Guenter Roeck, Pat Thaler and Manoj Wadekar.

References

[1] Dina Katabi, Mark Handley, and Chalrie Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM Sigcomm*, 2002.
[2] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen and Nick McKeown , "Processor Sharing Flows in the Internet," *Thirteenth International Workshop on Quality of Service (IWQoS)*, Passau, Germany, June 2005.
[3] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, architecture, algorithms, and performance," in *Proceedings of IEEE Infocom*, Mar. 2004.
[4] S. Athuraliya, S. Low, V. Li, and Q. Yin, "REM active queue management," *IEEE Network Mag.*, vol. 15, pp. 48-53, May 2001.
[5] Chris Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proceedings of IEEE Infocom*, April 2001.
[6] F. P. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237-252, Mar. 1998.
[7] Steven H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Transactions on Networking*, v.11 n.4, p.525-536, August 2003.
[8] F. P. Kelly, "Fairness and stability of end-to-end congestion control," *Eur. J. Control*, vol. 9, pp. 159-176, 2003.
[9] Y. Lu, R. Pan, B. Prabhakar, D. Bergamasco, V. Alaria and A. Baldini, "Congestion control in networks with no congestion drops," *Allerton*, September 2006.
[10] A. Varga, "OMNET++ Discrete Event Simulation," System User Manual. 2006.
[11] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC Editor, 2003.
[12] Rayadurgam Srikant, "The Mathematics of Internet Congestion Control (Systems and Control: Foundations and Applications)," SpringerVerlag, 2004.
[13] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle, "Dynamics of TCP/RED and a Scalable Control," in *Proceedings of IEEE Infocom*, vol. 1 pp. 239-248. New York, NY, 23-27 Jun 2002.
[14] Tom Kelly, "Scalable TCP: improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, v.33 n.2, April 2003.
[15] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM 2004*, pp. 2514-2524.
[16] I. Rhee and L. Xu., "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *PFLDNet'05*, February 2005.
[17] http://www.ieee802.org/1/pages/dcbridges.html
[18] http://www.ieee802.org/1/files/public/docs2008/au-prabhakar-qcn-los-gatos-0108.pdf
[19] http://www.ieee802.org/1/files/public/docs2008/au-rong-qcn-serial-hai-pseudo-code
[20] http://www.ieee802.org/1/files/public/docs2008/au-pan-qcn-benchmark-sims-0108.pdf
[21] Network Simulator. ns2. http://www.isi.edu/nsnam/ns.