Ethernet Congestion Manager

Davide Bergamasco

Cisco Systems, Inc.

Abstract

This document describes a congestion management mechanism aiming at controlling congestion in short-range, high-speed Ethernet networks such as Data Center Networks. Such mechanism, called Ethernet Congestion Manager (ECM), includes three components: (1) congestion detectors associated with bridge transmission queues, (2) rate limiters associated with NICs transmission queues to control the traffic injection rate, and (3) a signaling protocol to convey congestion control information from detectors to rate limiters. When congestion arises in some queue in the network, congestion control signals are sent from such queue to the NICs originating the flows causing congestion. Such signals will cause rate limiters to slow down the offending flows to a rate compatible with the transmission rate from the congested queue, effectively bringing down the queue depth to a desired level.

Acknowledgemtns: Many thanks to Bruce Kwan (Broadcom), Mitch Gusat (IBM), and Manoj Wadekar (Intel) for their kind contribution to this document.

Modification History

Revision	Date	Originator	Comments
0.1	2/5/07	Davide Bergamasco	Initial Draft

Table of Contents

List of Figures

1 Introduction

Data Center networks are a peculiar environment because of their high speed (at least 1 Gbps) and low latency (a few tens of microseconds of round trip time). In certain cases, such networks may make use of 802.3X link-level flow control to deliver near-zero packet loss to applications. Such requirements make congestion management in data center networks quite challenging because:

- High speed coupled with small buffers (required to provide low latency) causes such buffers to fill up extremely quickly when congestion arises;
- If link-level flow-control is being used, congestion spreads over a saturation tree almost instantly causing severe head of the line blocking, possibly live-lock, or – in the worst of scenarios – even dead-lock.

Traditional congestion control techniques such as RED [1] and ECN [2] have been shown not to work well with small buffers because of the extremely compressed dynamics exhibited by such buffers. In fact, under congestion conditions a buffer in a typical data center network may fill up in a few hundreds of microseconds, forcing RED and ECN to work in the region of maximum drop/mark probability. This, in turn, causes the traffic flows to back off too much, and – consequently – substantial loss of throughput. Also, RED and ECN are layer 3 and 4 congestion management mechanisms which work only in presence of a cooperating transport protocols such as TCP. Since in data center network there is a substantial presence of non-TCP traffic RED and ECN are ineffective at controlling congestion caused by such traffic.

To overcome the above limitations, we propose *Ethernet Congestion Manager* (ECM), a layer 2 congestion control mechanism conceived to operate in networks limited in scope as per the IEEE P802.1Qau Project Authorization Request (PAR) [3]. The founding principles of ECM are:

- Pushing congestion from the core of the network towards the edge, where there is less traffic aggregation and more resources to deal with it more effectively;
- Using rate-limiters at the edge to control the rate of traffic injection for flows causing congestion;
- Tuning rate-limiter parameters based on continuous feedback coming from the congestion points.

The rest of this document is organized as follows: section 2 provides an overview of the ECM mechanism; section 3 describes ECM in detail, particularly section 3.1 discusses the signaling component and the underlying protocol, section 3.2 addresses the congestion detection component, and section 3.3 examines the reaction component. Finally, conclusions are presented in section 4.

2 ECM Overview

Figure 1shows a sample data center network composed of a core switch, a number of edge switches and some end nodes. End Nodes A and B are simultaneously sending traffic at line rate (10 Gbps) to end node C. Since the aggregate traffic rate exceeds the capacity of the link connecting the Core Switch to Edge Switch C, this link is subject to congestion and the queue(s) associated with it start filling up. The *detection* component of ECM associated with that queue samples arriving frames with a certain probability. Based on a configurable threshold that define the ideal queue depth, if the current queue length exceeds the threshold, ECM *signals* “slow-down" by sending messages destined to end nodes A and B. Such messages are processed at end nodes A and B. The *reaction* to a "slow-down" message is the instantiation of a rate limiter (or a further slow down if one is already instantiated) at the processing point. The purpose of the rate limiter is to slow down a congesting traffic flow to mitigate congestion at the core switch.

Frames that have been rate limited carry this information with them. ECM continues to monitor the queue length and, as the congestion begins to abate, signals the nodes which have rate limited their traffic "speed up". This is to avoid under-utilizing the bandwidth at the congestion point.

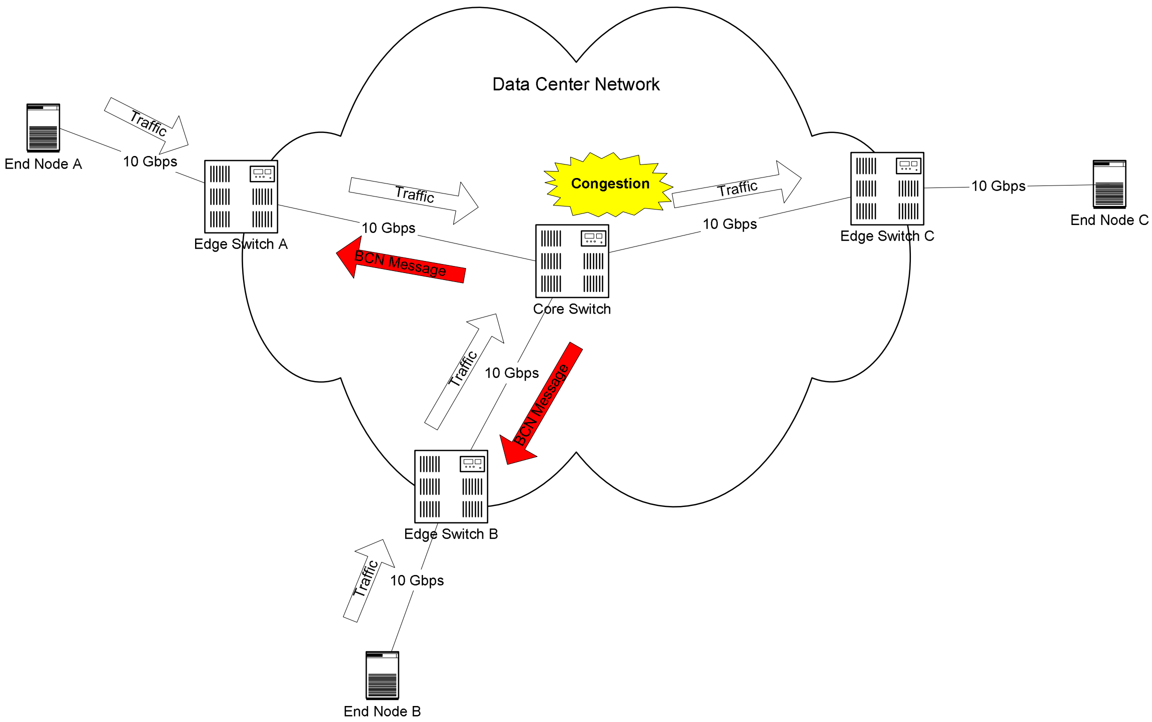

Figure 1 – Example of a congested Data Center Network and ECM messaging

2.1 Terminology

This section provides the definition of a number of terms that will be used throughout this document.

- ***Congestion Management Domain***: The *contiguous* set of Layer 2 devices that support ECM. In a given Congestion Management Domain (CMD), ECM may be enabled or disabled on each of the eight IEEE 802.1Q priorities independently.
- ***Congestion Point***: Place where an uncontrolled accumulation of data frames occurs because of the mismatch in the arrival rate and the departure rate. An example of congestion point (CP) is one of the output queues of a switch.
- ***Detection***: Component of ECM residing at a CP which detects a congestion condition and generates "slow-down" (or "speed-up") signals to bring such condition under control.
- ***Reaction Point***: Place where signals generated by the ECM detection component are 8processed and terminated. Reaction points (RPs) are usually located in Network Interface

Cards (NICs) and consist of a set of queues and an equal number of rate limiters associated with them.

- ***Reaction***: Component of ECM residing in an RP which implements congestion mitigation actions. The reaction component process ECM signals and accordingly control the rate of injection of traffic by adjusting the current rate of the rate limiters.
- ***Signaling***: Component of ECM used to carry congestion control messages from the CPs to the RPs.

The next section describes in detail the three components of the ECM mechanism, namely signaling, detection, and reaction.

3 ECM Components and Operations

3.1 Signaling

Figure 2 shows the exchange of messages between a CP and a RP. As soon as a CP detects congestion, as described later in section 3.2, it starts sending explicit feedback messages to the RPs associated with the traffic flows causing such congestion. The feedback message is an Ethernet frame known as the *ECM Frame*. A possible format for the ECM Frame is shown in Figure 3.

An ECM Frame is generated by a CP by sampling incoming frames, as described in section 3.2. The ECM Frame has *Destination Address* (DA) equal to the Source Address of the sampled frame, and a *Source Address* (SA) equal to a MAC address associated with the CP (usually the MAC address of the Management Entity of the switch where the CP is located).

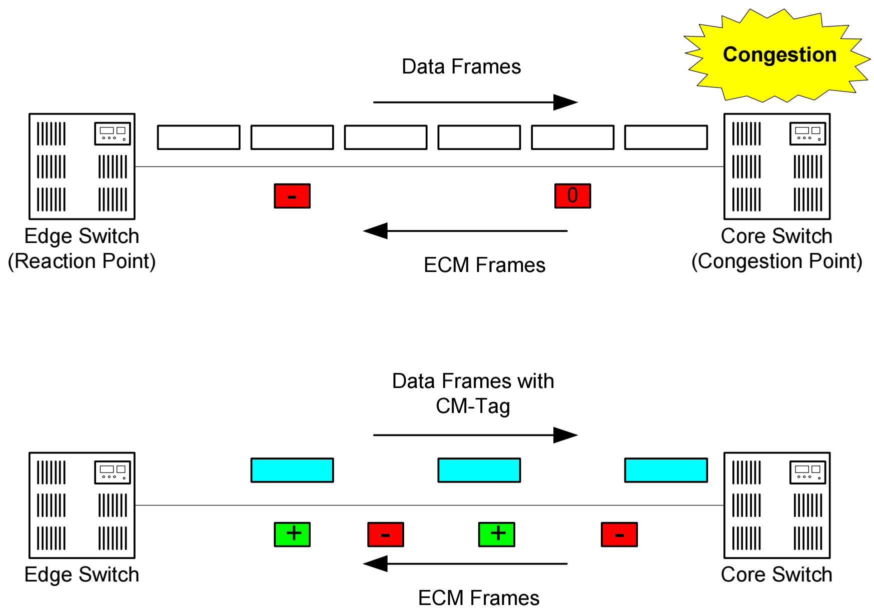

Figure 2 –ECM signaling

```
 0                             15                              31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+    DA = SA of sampled frame   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+     SA = MAC Address of CP    +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     IEEE 802.1Q Tag                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         EtherType = ECM       |Version|Q|      Reserved       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                           CPID                                +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Qoff               |            Qdelta             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Timestamp       |Unit |           Reserved              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|        First N bytes of sampled frame starting from DA        |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              FCS                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3 – ECM Frame Format

This ensures that the ECM Frame is forwarded back to the source of the traffic causing congestion with a valid source address.

The *IEEE 802.1Q Tag* is copied from the sampled frame. The Priority field of the ECM Frame 802.1Q Tag is set either to the priority of the sampled frame or to a configurable priority. It is preferable to use the highest priority in order to minimize the latency experienced by ECM Frames.

The *EtherType* of the ECM Frame is set to 0xXXXX, identifying the frame as being an ECM Feedback message.

The *Version* field indicates the version of the ECM protocol. Currently only version 0 (zero) is defined. Subsequent versions of the ECM protocol must be backward compatible. If a ECM implementation version X receives a ECM frame with version Y and Y > X, such an implementation must use only the fields defined for version X.

The *Q* bit indicates that the *Qdelta* field has saturated, i.e., its value is either equal to *-2Qeq* or *2Qeq*. The meaning of such bits will be clarified in section 3.3.1.

The bits in the *Reserved* fields are currently not used. They must be set to 0 (zero) on transmission and ignored on reception. Future versions of the BCN protocol may redefine all or some of the reserved bits.

The *CPID* field is the *Congestion Point IDentifier* and its purpose is to univocally identify a congested entity – usually a queue – within CMD. This information has to be propagated to the RP in order to create a bi-univocal association between the CP and RP(s). The CPID field must be unique across the network but, since it is an opaque object, its format is only relevant to the CP that assigns it. The CPID should at least include a MAC address associated with the switch where the CP resides to ensure global uniqueness, plus a local identifier to ensure local uniqueness.

The *Qoff* and *Qdelta* fields contain the actual feedback information conveyed by the Congestion Point to the Reaction Point. The use of such fields will be described in the next two sections.

The *Timestamp* and *Unit* fields are copied from the corresponding fields from CM-Tag (see below for its definition) of the sampled frame. If the sampled frame does not carry such a tag, the *Timestamp* and *Unit* fields are set to 0 (zero).

The payload of a BCN Frame consists of the first *N* bytes (where *N* is a configurable parameter) of the sampled frame starting from the DA. The minimum value of *N* is 24, to ensure that a BCN Frame is at least 64 bytes long. The purpose of such payload is to convey to the RP enough information to exert the finest congestion mitigation action possible (it should at least include DA, SA, and 802.1Q Tag).

When a RP receives an ECM Frame from a CP, and such message causes a congestion mitigation action to be performed on a particular traffic flow (usually the activation of a rate limiter or the adjustment of an existing one), the CPID field from the ECM Frame is stored in a local register associated with the corresponding rate limiter. All the frames belonging to that flow subsequently injected by the RP in the network will carry a *Congestion Management Tag* (CM-Tag) containing the CPID from the register (see section 3.3).

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        EtherType = CM-Tag     |Version|      Reserved         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                             CPID                              +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Timestamp       |Unit |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4 – Format of the CM-Tag

The CM-Tag is identified by the value 0xXXXX in the *EtherType* field and it should be located after the 802.1Q Tag. Its main purpose is to complete the bi-univocal association between a CP and a RP. The purpose of this association is to prevent a RP from receiving positive feedback from multiple CPs for the same flow. In fact, a CP is supposed to generate positive ECM Feedback messages only for frames that carry a CM-Tag with a CPID matching its own ID.

The *Version* field indicates the version of the ECM protocol. Currently only version 0 (zero) is defined. Subsequent version of the ECM protocol must be backward compatible. If a ECM implementation version X receives a ECM frame with version Y and Y > X, such an implementation must use only the fields defined for version X.

The *Timestamp* field is used by a RP to estimate the round trip time from the CP it is associated with. Every time a RP inserts a CM-Tag in the frame it is going to transmit, the current value of a local free running timer is copied into the Timestamp field. The *Unit* field indicates the time unit used by the free running timer according to the following equation:

$$\text{Time\_unit} = 2^{Unit} \cdot 1\ \mu\text{s}$$

3.2 Detection

Figure 5 shows how the detection process works and how ECM Frames are generated at a Congestion Point.

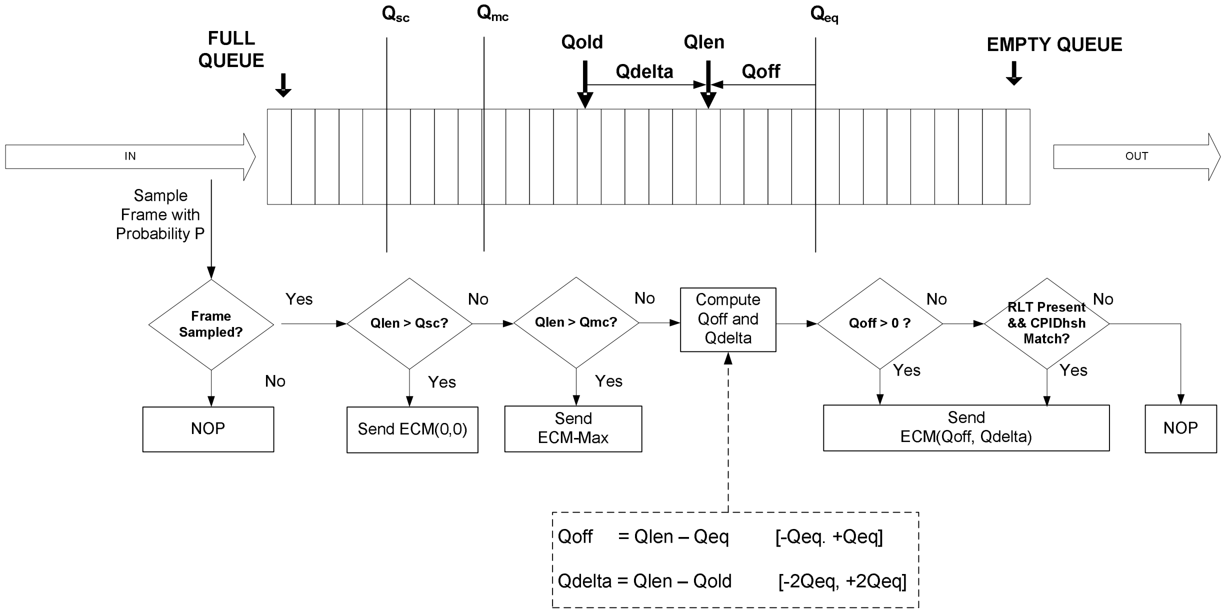

Figure 5 - Congestion Detection Process and Message Generation at a Congestion Point

An *equilibrium* threshold *Qeq* defines the operating point of a queue under congestion conditions. In other words, *Qeq* is the target level around which the queue length should oscillate under normal congestion conditions. A severe congestion threshold *Qsc* defines the level at which the queue is subject to extreme congestion conditions.

Incoming frames are sampled with a certain probability P, e.g., 0.01. Sampling is performed on a byte arrival basis. That is, if the average frame length is $E[L]$, then a frame is sampled on average every $E[L]/P$ byte received. If we assume an average frame length of 1000 bytes, then the average sampling rate is going to be one frame every 100 KB of data received. This can be easily implemented using a *Fixed Interval* followed by a *Random Interval* as shown in Figure 6.

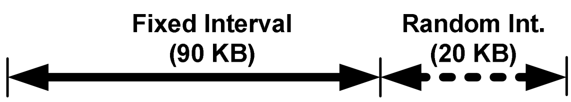

Figure 6 – Sampling Process

Initially and after every sample, the effective sampling interval S is calculated by adding the two intervals:

$$S = Sf + Sr$$

The length of every frame arriving at the queue is accumulated in *L*. The frame that makes *L* >= *S* is sampled. A new random interval *Sr* is picked and *L* is set to zero. The fixed interval *Sf* should be configurable in the range [0, 256] KB with 1 byte increments. The random interval *Sr* should be generated in the range [0, 64] KB with 1 byte increments.

When the queue length is zero (0) or it is above the *mild congestion threshold Qmc,* the sampling probability is increased by a factor *Sscale,* i.e., *S* is divided by *Sscale*[1]. This is called *over-sampling* and it's purpose is to speedup the response to transient conditions (i.e., queue going full or empty) by generating more ECM frames when such conditions occur.

When a frame is sampled, the current queue length *Qlen* is compared with the *Qsc* threshold. If *Qlen* is greater than *Qsc*, the queue is under severe congestion conditions, and a special ECM message, i.e., ECM(0,0), is generated. As discussed later in section 3.3, such message causes a rate limiter to temporarily drop its rate to zero.

If *Qlen* is below *Qsc* but above *Qmc*, the ECM message corresponding to the maximum negative feedback, i.e., ECM(Qeq, 2*Qeq) is generated. This message, also known as *ECM-Max* will cause the maximum rate decrement to occur at a rate limiter receiving it. ECM-Max, along with over-sampling, allows for a faster response to sudden and quick positive changes in the queue length.

If the queue is not operating under severe or mild congestion conditions, the two components of the ECM feedback, *Qoff* and *Qdelta*, are computed. As shown in Figure 5, *Qoff* is the offset of the current queue length with respect to the equilibrium threshold *Qeq*. *Qoff* must be saturated at +*Qeq* and –*Qeq*. *Qdelta* is the change in length of the queue since the last sampled frame, and it must be saturated at +2*Qeq* and -2*Qeq*. When *Qdelta* saturates, the *Q* bit in the ECM Frame is set. The unit of *Qeq*, *Qoff*, and *Qdelta* is multiples of 64 bytes.

If *Qoff* is positive, i.e., the queue is above the equilibrium threshold, a ECM message containing *Qoff* and *Qdelta* is generated. If this is not the case, a ECM message has to be generated only if the CM-Tag of the sampled frame contains an RL option and the CPIDhsh field matches the CPIDhsh associated with the queue. The rationale behind this is the following: If *Qoff* is positive, the queue is above the equilibrium and therefore an ECM message has to be generated anyway. In the other cases, the queue is either emptying out, or it is filling up but it has not yet reached the equilibrium threshold. In such cases, an ECM message has to be generated only on those flows that are currently rate limited and associated with this particular CP. This check is necessary to reduce as much as possible the generation of "false positive" ECM messages, i.e., positive ECM messages for non rate-limited flows, or for rate limited flows associated with other CPs.

In certain networks *Qeq* may be set differently for different CPs. In order to generate ECM messages carrying a normalized feedback, a scaling factor *Qscale* is used to multiply the values of *Qoff* and *Qdelta* copied into an ECM frame. In other words, *Qoff* and *Qdelta* are calculated as described above and the actual ECM frame will contain *Qscale·Qoff* and *Qscale·Qdelta*[2]. For example, a CP with a smaller buffer than other CPs may have a lower Qeq. This will result in a smaller range for the *Qoff* and *Qdelta* values generated by such CP compared with other CPs. To compensate for that, such CP will use a *Qscale* value greater than one.

[1] From the implementation perspective such division can be safely approximated with a shift operation as, most likely, *Sscale* is a power of 2.

[2] From the implementation perspective such multiplications can be safely approximated with shift operations as, most likely, *Qscale* is a power of 2.

When *Qoff* and *Qdelta* are both zero, no ECM message is generated. As described earlier, the message ECM(0,0) has a special meaning.

Every ECM frame generated by a CP will carry in the payload the CM-Tag copied from the sampled frame. This information is used by the RP as described in the next section.

3.2.1 Congestion Point Pseudo-code

```
initialize()
{
    queue_len = 0;       // Queue length. Incremented on packet arrival by packet length (in
                         // pages) and decremented on packet departure.
    samp_byte_acc = 0;   // Sampling process byte arrival accumulator.
    queue_old = 0;       // Queue length at previous sample.

    ecm_sampling_interval = ECM_FIXED_SAMPLING_INT +
              (ECM_RANDOM_SAMPLING_INT * (urand() * 2 - 1));    // urand(): random number
                                                                 // uniformly distributed in [0,1)
}


foreach ( IncomingFrame = frame_arrival() )
{
    if ( Enable_ecm_generation &&
         IncomingFrame.Ethertype != ECM )            // Sample only frames subject to ECM
    {
        samp_byte_acc += len( IncomingFrame );
        if ( samp_byte_acc > ( ecm_sampling_interval >>
                               ((queue_len > ECM_Q_MC || queue_len == 0) ? ECM_S_SCALE : 0 )))
        {
            /* Frame has been sampled */
            need_gen_ecm_frame = 1;                  // Assume an ECM frame has to be generated

            /* Setup next sampling interval */
            samp_byte_acc = 0;
            ecm_sampling_interval = ECM_FIXED_SAMPLING_INT +
                                   (ECM_RANDOM_SAMPLING_INT * (urand() * 2 - 1));

            if ( queue_len > ECM_Q_SC )              // ECM(0,0)
            {
                ECMFrame.Qoff = 0;
                ECMFrame.Qdelta = 0;
            }
            else if ( queue_len > ECM_Q_MC )         // ECM-Max
            {
                ECMFrame.Qoff = ECM_Q_EQ;
                ECMFrame.Qdelta = 2 * ECM_Q_EQ;
            }
            else                                     // Regular ECM Frame
            {
                qoff = queue_len - ECM_Q_EQ;
                if (qoff < -ECM_Q_EQ) qoff = -ECM_Q_EQ;
                ECMFrame.Qoff = qoff * ECM_Q_SCALE;

                qdelta = queue_len - queue_old;
                if ( qdelta > 2 * ECM_Q_EQ )
                {
                    qdelta = 2 * ECM_Q_EQ;
                    if ( ECM_qsat_enable ) ECMFrame.Q = 1;   // NB: experimental feature,
                                                             // enable needed
                }
                if ( qdelta < -2 * ECM_Q_EQ )
                {
                    qdelta = -2 * ECM_Q_EQ;
                    if ( ECM_qsat_enable ) ECMFrame.Q = 1;
                }
                ECMFrame.Qdelta = qdelta * ECM_Q_SCALE;
```

```
            /* Filter out spurious feedback */
            if ((qoff == 0 && qdelta == 0) ||   // No rate change
                ((qoff < 0) &&                  // Positive Fb and …
                 (!has_cmtag(SampledFrame) ||  // … no CM-Tag or CPID mismatch
                  (has_cmtag(SampledFrame) && SampledFrame.CMTag.CPID != CPID))))
            {
                Need_gen_ecm_frame = 0;
            }
        }
        queue_old = queue_len;

        if ( NeedGenECMFrame )
        {
            ECMFrame.DA = IncomingFrame.SA;
            ECMFrame.SA = SWITCH_MAC_ADDRESS;        // MAC address of the switch generating
                                                     // ECM frame
            ECMFrame.8021QTag.Priority = HIGH_PRI;   // May be priority of sampled frame
            ECMFrame.CPID = CPID;
            ECMFrame.Timestamp = SampledFrame.CMTag.Timestamp;
            ECMFrame.Unit = SampledFrame.CMTag.Unit;
            forward( ECMFrame );
        }
    }    // if (samp_byte_acc >= ECM_sampling_interval )
  }    //  if ( Enable_ecm_generation …
}   // foreach()
```

3.3 Reaction

Figure 7 shows the structure of an RP data-path which may be implemented in the egress port of NIC. A set of filters, F1 through F*n*, divert the traffic that matches a particular filtering criterion (e.g., {VLAN, DA, SA, Priority}, {VLAN, Priority}, {Priority}, etc.) from the regular data path ("No Match" in the figure) to a set of corresponding queues. Traffic is drained from such queues by a set of rate limiters, R1 through R*n*, whose rate is controlled by the ECM Frames coming from CP. Note that, in order to avoid out-of-order frames, the "No Match" path must not queue traffic, i.e., it has absolute priority with respect to the rate-limited paths. Besides controlling the rate of traffic, the rate limiters also add the CM-Tag to all the transmitted frames in order to elicit feedback from the CPs they are currently associated with.

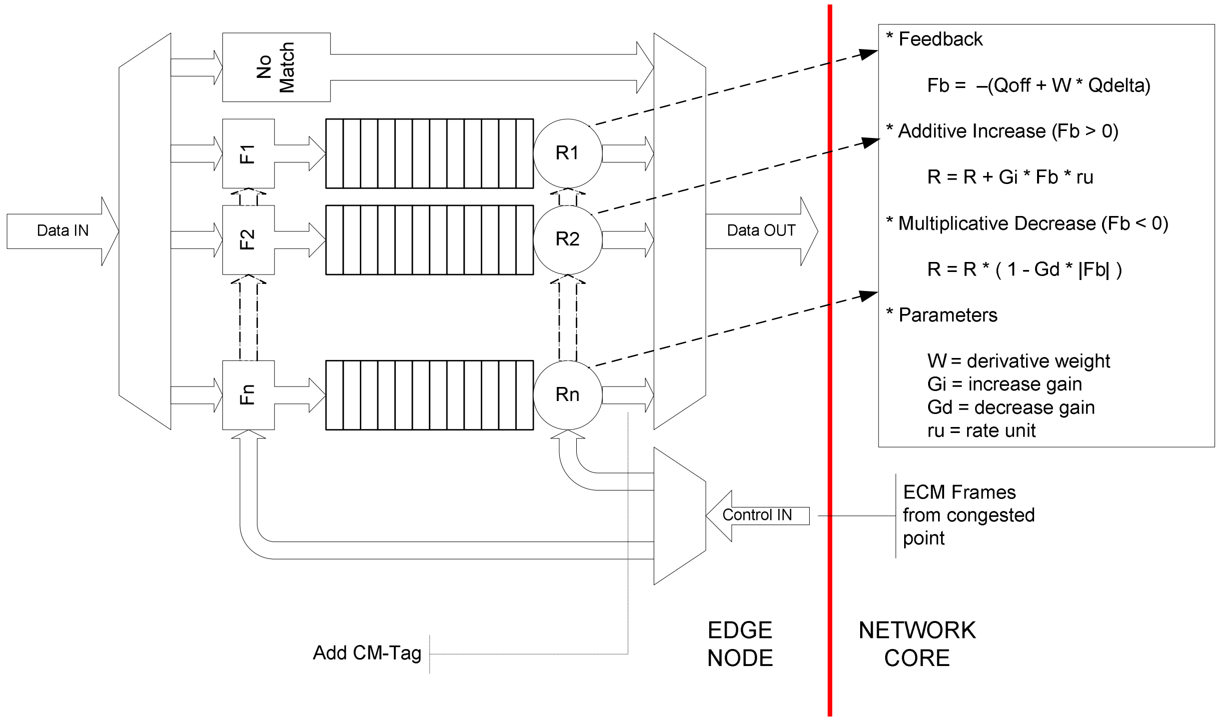

Figure 7 – Data-path structure of a Reaction Point

Every time an ECM frame is received by an RP, the information necessary to identify a traffic flow (e.g., DA, SA, VLAN, Priority) is extracted from the header of the sampled frame carried inside the ECM frame. This information is combined in some way (for example using a hash function) to obtain a *compressed flow identifier* (*FlowID*). The FlowID is compared with the FlowIDs stored in the currently active filters. If there is no match, this flow is not currently being rate limited by the RP. A filter/rate-limiter pair is instantiated and the FlowID and CPID received in the ECM frame are stored in registers associated with this pair. Also, the rate of such rate limiter is set to a configurable initial rate *Ri*[3].

If there is a match, however, this particular flow is already being rate limited by the RP. The value of the feedback from the CP is then calculated as described later in this section. If the feedback is negative, the rate-limiter rate is adjusted as per feedback, and CPID from the ECM frame are stored with the filter/rate-limiter pair. If the feedback is positive, instead, the rate is adjusted if and only if the CPID of the ECM frames matches the CPID currently stored with the rate-limiter. In other words, all ECM frames carrying a negative feedback are honored, while the ECM frames carrying a positive feedback are processed only if they have been generated by the CP currently associated with the RP.

Since FlowID is a compressed flow identifier, multiple flows may be identified the same FlowID and end up in the same rate-limited queue. Different implementations may choose the degree of flow aggregation by using a different number of rate-limited queues (*n*) and a different hash function to obtain FlowID.

[3] Usually a fraction of the link capacity *C* such as ½ or ¼.

To ensure that (potentially) positive feedback is generated only by the CP currently associated with a filter/rate-limiter, the RP adds the CM-Tag to the frames it transmits containing the CPID currently stored in the rate-limiter register.

An active filter may change its association with a CP over time. As mentioned above, the association can be changed only when a ECM frame conveying negative feedback is received from a CP different from the one currently associated with the filter. For example, if a traffic flow is subject to congestion at CP1 – and, therefore, is rate controlled by CP1 – starts experiencing congestion at CP2, the latter will generate negative ECM frames for that flow, causing its filter to change association from CP1 to CP2. After some time, the negative feedback generated by one of the two Congestion Points will prevail over the other and the filter will settle its association with the prevailing one.

Every time an RP receives an ECM frame, the *Round Trip Time* (RTT) between the RP and the CP (defined as the difference between the current time and the Timestamp field carried by the ECM Frame) is calculated. The RTT is then filtered through an *exponential weighted moving average* (EWMA) as follows:

$$RTTavg \leftarrow (1 - 2^{-Wrtt})\ RTTavg + 2^{-Wrtt} * RTT$$

and made available to control software through a read-only register. This measure may be used to dynamically adjust the value of some of the control loop parameters. The algorithm to carry out such an adjustment is still being investigated.

At every RP there are only a limited number (n) of rate limiters available. Thus, it may happen that, at a certain moment, all the rate limiters are in use and an ECM frame arrives that would cause the instantiation of a new rate limiter. In such a case, a per-priority rate limiter (a.k.a. *coarse rate limiter*) gets instantiated, and all the rate limiters (a.k.a. *fine rate limiters*) currently associated with such a priority are set to operate at the maximum rate to force a rapid transition to the per-priority rate-control. When all the queues of the individual rate-limiters are empty, they may be released.

Once a rate limiter has been instantiated, it may be reclaimed once two conditions are satisfied: (1) the queue of the rate limiter is empty, and (2) its rate is at or above line-rate. These two conditions are necessary to avoid out of order packet delivery.

3.3.1 Rate Control Algorithm

The rate control algorithm employed by ECM works according to an *Additive Increase Multiplicative Decrease* (AIMD) scheme loosely derived from the one employed by TCP. TCP increases its rate linearly over time in absence of congestion and halves its rate every time it receives negative feedback, either explicit (i.e., ECN), or implicit (i.e., packet drop). The granularity of this AIMD scheme is quite coarse and it has been shown that in many cases it may lead to link underutilization. In contrast, ECM employs an AIMD scheme with a much finer granularity. Every time a ECM Frames arrives at a rate limiter, a Feedback signal is calculated according to the following equation:

$$Fb = -(Qoff + w \cdot Qdelta)$$

where w is a parameter used to weight the delta component more or less with respect to the offset component . w should be configurable in the interval [1/8, 8] with 1/8 increments. Based on the sign of the Feedback signal Fb, the rate is increased or decreased as follows:

- If $Fb > 0$ $\quad R \leftarrow R + Gi \cdot Fb \cdot Ru$
- If $Fb < 0$ $\quad R \leftarrow R \cdot (1 - Gd \cdot |Fb|)$

where Gi and Gd are the *Increase Gain* and *Decrease Gain* respectively, and Ru is the *Rate Unit* (i.e., the granularity of the rate adjustment) employed by the rate limiters. Ru should be configurable in the range [1, 100] Mbps with increments of 1 Mbps. Both Gi and Gd are fractional values. Since their granularity is unknown, they should be represented in fixed point notation with the largest number of bits possible.

Since Gd and Qeq may be chosen independently, to limit the maximum negative rate adjustment to a fraction $\alpha < 1$, the product $Gd \cdot |Fb|$ must be saturated to α. Likewise, since Gi and Qeq may be independently chosen, the maximum positive rate adjustment should be limited to a certain fraction $\beta < 1$ of the link capacity C[4]. Therefore the product $Gi \cdot Fb \cdot Ru$ must be saturated to $\beta \cdot C$. Given such constrains, the previous equations can be rewritten as:

- If $Fb > 0$ $\quad R \leftarrow R + min(Gi \cdot Fb \cdot Ru, \beta \cdot C)$
- If $Fb < 0$ $\quad R \leftarrow R \cdot (1 - min(Gd \cdot |Fb|, \alpha))$

Just like Gi and Gd, α and β are fractions and they should be represented in fixed point notation with as many bits as possible.

Besides the changes driven by feedback from CPs, the current rate of a rate limiter is also subject to a periodical *self-increase*. Every time interval T_d (e.g., 1 ms), the rate is increased by a small amount R_d (e.g., 1 Mbps). Such self-increase is useful for a number of reasons:

1. Speeds up convergence to fairness, as small flows receive substantially larger relative increments compared with large flows;
2. Allows for the reclamation of stale rate limiters. In fact, a rate limiter may stop receiving ECM frames because two main reasons: (1) the traffic stream that such rate limiter was controlling has suddenly ended, and (2) routing issues in the network prevent ECM Frames from reaching the rate limiter. When this happens, the rate limiter will remain stuck at the current rate forever. Instead, the self-increase will bring the rate-limiter rate back to line-rate and will cause its decommissioning;
3. Improves the recovery after a ECM(0,0) message is received (see next subsection).

T_d should be programmable in the range [1 μs, 10 ms] with 1 μs increments, while R_d should be programmable in the range [1 Mbps, 100 Mbps] with 1 Mbps increments.

Different (and more complex) self-increase strategies may be employed by a reaction point (see [7]). The one described in this document has been chosen for its simplicity.

---

[4] C is the capacity of the link draining the rate limiter.

3.3.2 Exceptions and Non-linear Rate Adjustments

When a CP is subject to severe congestion, it may send the special ECM frame ECM(0,0) (i.e., a ECM message with *Qoff* =0 and *Qdelta* = 0). When a rate limiter receives such a message, as shown in Figure 8, it sets its current rate *R* to 0 and starts a random timer whose range is determined by a parameter *Tmax* (e.g., 10 µs). When the timer started by the ECM(0,0) frame expires, the rate limiter is set to operate at a minimum rate *Rmin* (e.g., 1/100 of line rate). This should restart the traffic flow towards the congestion point and trigger – hopefully positive – feedback. During the random timeout period the automatic self-increase of the rate is suspended, and it is resumed only after the timer expiration. Also, all ECM messages, including ECM(0,0) must be ignored during a timeout period.

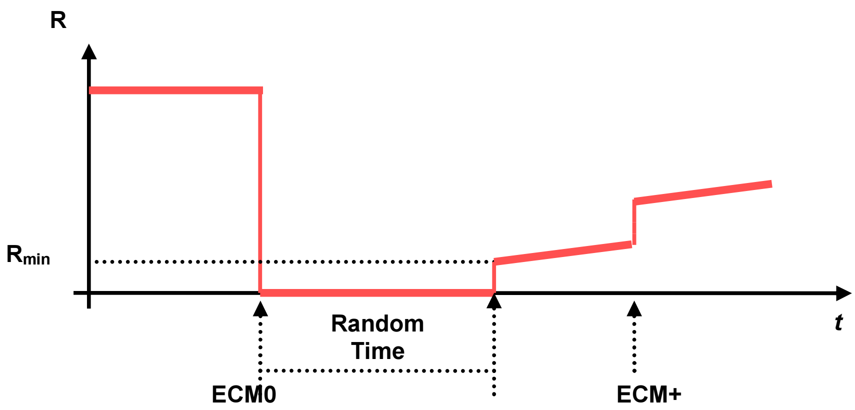

Figure 8 – Example of timeout and random restart

After the timer expiration, *Tmax* is doubled and *Rmin* is halved, so that the next ECM(0,0) (if any) will cause the random timer to have a longer duration and the rate limiter to restart from a slower rate, effectively realizing an exponential back-off. The initial values of *Tmax* and *Rmin* are restored upon the reception of the first positive feedback.

Tmax should be configurable in the range [1µs, 1s] with 1µs increments, while *Rmin* should be configurable in the range [1 Mbps, 1 Gbps] with 1 Mbps increments. The initial values of *Tmax* and *Rmin* are 10 us and 100 Mbps respectively.

The timeout with random restart has been introduced with the goal to mimic two behaviors which have been proven very successful when dealing with severe congestion caused by multiple traffic sources:

1. TCP retransmission timeout, which causes the silencing of most of the sources contributing to congestion, and
2. Ethernet CSMA/CD algorithm, which helps desynchronize traffic sources restarting after a timeout.

Special handling of ECM messages is also required when the *Q* is set in the ECM Frame, signaling that the *Qdelta* feedback component is saturated at *2Qeq* or -*2Qeq*. When this happens, a stronger rate adjustment must be performed because the system is working outside of the linear region. The following rate adjustment is performed based on the sign of *Qdelta*:

- If *Qdelta* < 0 $R \leftarrow R + 2\cdot\beta\cdot C$
- If *Qdelta* > 0 $R \leftarrow R\cdot(1 - min(2\cdot\alpha, 1))$

In other words, when *Qdelta* saturates a rate adjustment twice as big as the maximum rate adjustment in either direction is performed. Since in the case of the decrease $2\cdot\alpha$ may be larger than 1, the resulting rate may be negative. To avoid this, the product $2\cdot\alpha$ is saturated at 1.

3.3.3 Reaction Point Pseudo-code

```
initialize()
{
    RL[*].state = INACTIVE;             // * = all rate limiters
    RL[*].flowid = -1;                  // no flow id
    RL[*].rate = C;
    RL[*].CPID = 0;
    RL[*].RTTavg = 0;
    RL[*].Rmin = Rmin;
    RL[*].Tmax = Tmax;
}


processECMFrame ( ECMFrame )
{
    FlowId = some_hash( ECMFrame.Payload );    // ECM frame payload contains the header of
                                               // sampled frame

    rlidx = getRateLimiterIndx( FlowID );      // Returns the index of the RL associated with
                                               // FlowID, or the index of the next available
                                               // RL if no FlowID match. Note that FlowID may
                                               // be used directly as the index in the RL table

    if ( ( ECMFrame.Qoff == 0 && ECMFrame.Qdelta == 0 ) &&    // ECM(0,0)
          RL[rlidx].state != TIMEOUT )
    {
        RL[rlidx].state = TIMEOUT;
        RL[rlidx].rate = 0;
        RL[rlidx].CPID = ECMFrame.CPID;
        Tmax_timer_set( rlidx, RL[rlidx].Tmax * urand() );
    }
    else
    {
        Fb = -(ECMFrame.Qoff + W * ECMFrame.Qdelta);

        if ( Fb < 0 )
        {
            if (RL[rlidx].state = INACTIVE )
            {
                RL[rlidx].state = ACTIVE;
                RL[rlidx].flowid = FlowId;
                RL[rlidx].rate = Ri;
                RL[rlidx].CPID = ECMFrame.CPID;
                RL[rlidx].RTTavg = 0;
            }
            else
            {
                RL[rlidx].rate *= 1 - ( ECMFrame.Q == 1 ? min(2·α, 1) : min(-Fb·Gd, α) ) ;
                if ( RL[rlidx].rate == 0 ) RL[rlidx].rate = RL[rlidx].Rmin;  // Saturate to Rmin
                if ( RL[rlidx].CPID == ECMFrame.CPID )
                {
                    RL[rlidx].RTTavg = calc_movavg( RL[rlidx].RTTavg,                  // old value
                                                    (now() - ECMFrame.Timestamp),      // new value
                                                    2 ^ -Wrtt );                       // weight
```

```
                }
                else
                {
                    RL[rlidx].CPID = ECMFrame.CPID;
                    RL[rlidx].RTTavg = 0;
                }
            }
        }
        else if ( RL[rlidx].CPID == ECMFrame.CPID &&
                  RL[rlidx].state == ACTIVE )
        {
            RL[rlidx].rate += ECMFrame.Q == 1 ? 2·β·C : min(Gi·Fb·Ru, β·C);
            RL[rlidx].Rmin = Rmin;
            RL[rlidx].Tmax = Tmax;
            RL[rlidx].RTTavg = calc_movavg( RL[rlidx].RTTavg,                // old value
                                            (now() - ECMFrame.Timestamp),    // new value
                                            2 ^ -Wrtt );                     // weight
        }
    }
}

/* Timers */

foreach ( rlidx = Tmax_timeout() )
{
    RL[rlidx].state = ACTIVE;
    RL[rlidx].rate = RL[rlidx].Rmin;
    RL[rlidx].Rmin /= 2;
    RL[rlidx].Tmax *= 2;
}


foreach ( Td_timeout() )
{
    if ( RL[*].state == ACTIVE )                  // * = all Rate Limiters
    {
        RL[*].rate += Rd;
        if ( RL[*].rate > C ) RL[*].rate = C;    // saturate @ C
    }
    Td_timer_set( Td );
}


/* Frame departure from RL queue */

foreach ( rldix = frame_departure_from_rl() )
{
    insertCMTag( OutgoingFrame );
    OutgoingFrame.CMTag.CPID = RL[rlidx].CPID;
    OutgoingFrame.CMTag.Timestamp = now();

    if ( RL[rlidx].queue_len == 0 && RL[rlidx].rate == C )
    {
        RL[rlidx].state = INACTIVE;
        RL[rlidx].flowid = -1;
    }
}
```

4 Conclusions

This document describes ECM, a backward notification-based mechanism for congestion management in data center networks. Such networks are peculiar because of their high speed, low latency, and, in certain cases, zero traffic loss. In such environments traditional congestion management mechanism such as RED [1] and ECN [2] have been shown not to work particularly well.

Simulation evidence [4] [5] shows that ECM works substantially better than the above mentioned alternatives, especially in data center networks where TCP and non-TCP traffic share the same infrastructure. This is because traditional congestion management schemes work only when the vast majority of traffic is TCP, i.e., they assume a congestion-responsive transport layer. Since ECM does not make any assumption on the transport layer, it can deal even with non-responsive protocols.

ECM has also been studied from the analytical standpoint using techniques commonly used in Control Theory. It has been analytically shown that the ECM control loop is stable in a wide region as determined by its parameters [6].

ECM has been originally presented to 802.1 in May 2005 for review and to gather feedback from the standards community[5] [4][5][6]. A Project Authorization Request (PAR) [1], along with a tutorial on ECM, was presented to the IEEE 802 Plenary in July 2006. The PAR was approved and a Task Force named 802.1Qau has been formed with the charter to develop a congestion management framework for Ethernet. ECM is currently one of the proposals being considered by the task force for such framework.

5 References

[1] Floyd, S., and Jacobson, V., "Random Early Detection gateways for Congestion Avoidance**",** V.1 N.4, August 1993, p. 397-413

[2] Ramakrishnan, K.K., Floyd, S., and Black, D.," The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168

[3] P802.1Qau (C/LM) Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks - Amendment 10: Congestion Notification

[4] D. Bergamasco, "Presentation of Backward Congestion Notification to IEEE 802.1", http://www.ieee802.org/1/files/public/docs2005/new-bergamasco-backward-congestion-notification-0505.pdf

[5] D. Bergamasco, "Updates on BCN For the IEEE 802 July 2005 Plenary Meeting", http://www.ieee802.org/1/files/public/docs2005/new-bergamasco-bcn-july-plenary-0705.ppt

[6] D. Bergamasco, R. Pan, "Presentation of BCN V2.0 to IEEE 802.1 Sep 2005 Interim Meeting", http://www.ieee802.org/1/files/public/docs2005/new-bergamasco-bcn-september-interim-rev-final-0905.ppt

[7] Y. Lu, R. Pang, B. Prabhakar, D. Bergamasco, V. Alaria, A. Baldini, "*Congestion Control in Networks with No Congestion Drops (I)*", 44th Annual Allerton Conference, September 2006

6 Glossary

The following list describes acronyms and definitions for terms used throughout this document:

AIMD: Additive Increase Multiplicative Decrease

[5] At that time, ECM was known as BCN, or *Backward Congestion Notification*. The name has been recently changed into ECM to avoid confusion with the generic concept of sending congestion notifications in the opposite direction of the traffic causing congestion.

AQM: Active Queue Management
CMD: Congestion Management Domain
CM-Tag: Congestion Management Tag
CP: Congestion Point
CPID: Congestion Point IDentifier
CSMA/CD: Carries Sense Multiple Access with Collision Detection
DA: Destination Address
ECM: Ethernet Congestion Management
ECN: Explicit Congestion Notification
FlowID: Flow Identifier
NIC: Network Interface Card
RED: Random Early Detection
RL: Rate Limiter
RL Option: Rate Limited Option
RP: Reaction Point
RTT: Round Trip Time
SA: Source Address
TCP: Transmission Control Protocol
VLAN: Virtual Local Area Network