Coupled congestion control for RTP media
draft-ietf-rmcat-coupled-cc-06

Abstract

   When multiple congestion controlled RTP sessions traverse the same
   network bottleneck, combining their controls can improve the total
   on-the-wire behavior in terms of delay, loss and fairness. This
   document describes such a method for flows that have the same sender,
   in a way that is as flexible and simple as possible while minimizing
   the amount of changes needed to existing RTP applications. It
   specifies how to apply the method for the NADA congestion control
   algorithm, and provides suggestions on how to apply it to other
   congestion control algorithms.

Table of Contents

1. Introduction

   When there is enough data to send, a congestion controller must
   increase its sending rate until the path's capacity has been reached;
   depending on the controller, sometimes the rate is increased further,
   until packets are ECN-marked or dropped. This process inevitably
   creates undesirable queuing delay when multiple congestion controlled
   connections traverse the same network bottleneck.

   The Congestion Manager (CM) [RFC3124] couples flows by providing a
   single congestion controller. It is hard to implement because it
   requires an additional congestion controller and removes all per-
   connection congestion control functionality, which is quite a
   significant change to existing RTP based applications. This document
   presents a method to combine the behavior of congestion control
   mechanisms that is easier to implement than the Congestion Manager
   [RFC3124] and also requires less significant changes to existing RTP
   based applications. It attempts to roughly approximate the CM
   behavior by sharing information between existing congestion
   controllers. It is able to honor user-specified priorities, which is
   required by rtcweb [RFC7478].

   The described mechanisms are believed safe to use, but are
   experimental and are presented for wider review and operational
   evaluation.

2. Definitions

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [RFC2119].

   Available Bandwidth:
        The available bandwidth is the nominal link capacity minus the
        amount of traffic that traversed the link during a certain time
        interval, divided by that time interval.

   Bottleneck:
        The first link with the smallest available bandwidth along the
        path between a sender and receiver.

   Flow:
        A flow is the entity that congestion control is operating on.
        It could, for example, be a transport layer connection, an RTP
        stream [RFC7656], whether or not this RTP stream is multiplexed
        onto an RTP session with other RTP streams.

   Flow Group Identifier (FGI):

A unique identifier for each subset of flows that is limited by
a common bottleneck.

Flow State Exchange (FSE):
The entity that maintains information that is exchanged between
flows.

Flow Group (FG):
A group of flows having the same FGI.

Shared Bottleneck Detection (SBD):
The entity that determines which flows traverse the same
bottleneck in the network, or the process of doing so.

3. Limitations

Sender-side only:
Coupled congestion control as described here only operates
inside a single host on the sender side. This is because,
irrespective of where the major decisions for congestion
control are taken, the sender of a flow needs to eventually
decide on the transmission rate. Additionally, the necessary
information about how much data an application can currently
send on a flow is often only available at the sender side,
making the sender an obvious choice for placement of the
elements and mechanisms described here.

Shared bottlenecks do not change quickly:
As per the definition above, a bottleneck depends on cross
traffic, and since such traffic can heavily fluctuate,
bottlenecks can change at a high frequency (e.g., there can be
oscillation between two or more links). This means that, when
flows are partially routed along different paths, they may
quickly change between sharing and not sharing a bottleneck.
For simplicity, here it is assumed that a shared bottleneck is
valid for a time interval that is significantly longer than the
interval at which congestion controllers operate. Note that,
for the only SBD mechanism defined in this document
(multiplexing on the same five-tuple), the notion of a shared
bottleneck stays correct even in the presence of fast traffic
fluctuations: since all flows that are assumed to share a
bottleneck are routed in the same way, if the bottleneck
changes, it will still be shared.

4. **Architectural overview**

   Figure 1 shows the elements of the architecture for coupled
   congestion control: the Flow State Exchange (FSE), Shared Bottleneck
   Detection (SBD) and Flows. The FSE is a storage element that can be
   implemented in two ways: active and passive. In the active version,
   it initiates communication with flows and SBD. However, in the
   passive version, it does not actively initiate communication with
   flows and SBD; its only active role is internal state maintenance
   (e.g., an implementation could use soft state to remove a flow's data
   after long periods of inactivity). Every time a flow's congestion
   control mechanism would normally update its sending rate, the flow
   instead updates information in the FSE and performs a query on the
   FSE, leading to a sending rate that can be different from what the
   congestion controller originally determined. Using information
   about/from the currently active flows, SBD updates the FSE with the
   correct Flow State Identifiers (FSIs). This document describes both
   active and passive versions, however the passive version is put into
   the appendix as it is extremely experimental. Figure 2 shows the
   interaction between flows and the FSE, using the variable names
   defined in Section 5.2.

```
                      -------   <---   Flow 1
                     | FSE |   <---   Flow 2 ..
                      -------   <---    .. Flow N
                        ^
                        |              |
                      -------          |
                     | SBD |   <-------|
                      -------
```

           Figure 1: Coupled congestion control architecture

```
   Flow#1(cc)                       FSE                 Flow#2(cc)
   ----------                       ---                 ----------
   #1 JOIN     ----register--> REGISTER

                                REGISTER     <--register-- JOIN #1

   #2 CC_R(1)  ----UPDATE----> UPDATE (in)

   #3 NEW RATE <---FSE_R(1)-- UPDATE (out) --FSE_R(2)-> #3 NEW RATE
```
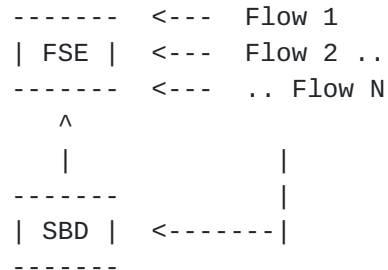
Figure 2: Flow-FSE interaction

Since everything shown in Figure 1 is assumed to operate on a single
host (the sender) only, this document only describes aspects that
have an influence on the resulting on-the-wire behavior. It does,
for instance, not define how many bits must be used to represent
FSIs, or in which way the entities communicate. Implementations can
take various forms: for instance, all the elements in the figure
could be implemented within a single application, thereby operating
on flows generated by that application only. Another alternative
could be to implement both the FSE and SBD together in a separate
process which different applications communicate with via some form
of Inter-Process Communication (IPC). Such an implementation would
extend the scope to flows generated by multiple applications. The
FSE and SBD could also be included in the Operating System kernel.

5. Roles

This section gives an overview of the roles of the elements of
coupled congestion control, and provides an example of how coupled
congestion control can operate.

5.1. SBD

SBD uses knowledge about the flows to determine which flows belong in
the same Flow Group (FG), and assigns FGIs accordingly. This
knowledge can be derived in three basic ways:

1. From multiplexing: it can be based on the simple assumption that
    packets sharing the same five-tuple (IP source and destination
    address, protocol, and transport layer port number pair) and
    having the same values for the Differentiated Services Code Point
    (DSCP) and the ECN field in the IP header are typically treated
    in the same way along the path. The latter method is the only
    one specified in this document: SBD MAY consider all flows that
    use the same five-tuple, DSCP and ECN field value to belong to
    the same FG. This classification applies to certain tunnels, or

RTP flows that are multiplexed over one transport (cf.
[transport-multiplex]). Such multiplexing is also a recommended
usage of RTP in rtcweb [rtcweb-rtp-usage].

2. Via configuration: e.g. by assuming that a common wireless uplink
    is also a shared bottleneck.

3. From measurements: e.g. by considering correlations among
    measured delay and loss as an indication of a shared bottleneck.

The methods above have some essential trade-offs: e.g., multiplexing
is a completely reliable measure, however it is limited in scope to
two end points (i.e., it cannot be applied to couple congestion
controllers of one sender talking to multiple receivers). A
measurement-based SBD mechanism is described in [I-D.ietf-rmcat-sbd].
Measurements can never be 100% reliable, in particular because they
are based on the past but applying coupled congestion control means
to make an assumption about the future; it is therefore recommended
to implement cautionary measures, e.g. by disabling coupled
congestion control if enabling it causes a significant increase in
delay and/or packet loss. Measurements also take time, which entails
a certain delay for turning on coupling (refer to
[I-D.ietf-rmcat-sbd] for details). Using system configuration to
decide about shared bottlenecks can be more efficient (faster to
obtain) than using measurements, but it relies on assumptions about
the network environment.

5.2. FSE

The FSE contains a list of all flows that have registered with it.
For each flow, it stores the following:

o   a unique flow number f to identify the flow.

o   the FGI of the FG that it belongs to (based on the definitions in
    this document, a flow has only one bottleneck, and can therefore
    be in only one FG).

o   a priority P(f), which is a positive number, greater than zero.

o   The rate used by the flow in bits per second, FSE_R(f).

Note that the absolute range of priorities does not matter: the
algorithm works with a flow's priority portion of the sum of all
priority values. For example, if there are two flows, flow 1 with
priority 1 and flow 2 with priority 2, the sum of the priorities is
3. Then, flow 1 will be assigned 1/3 of the aggregate sending rate
and flow 2 will be assigned 2/3 of the aggregate sending rate.

Priorities can be mapped to the "very-low", "low", "medium" or "high"
priority levels described in [I-D.ietf-rtcweb-transports] by simply
using the values 1, 2, 4 and 8, respectively.

In the FSE, each FG contains one static variable S_CR which is the
sum of the calculated rates of all flows in the same FG. This value
is used to calculate the sending rate.

The information listed here is enough to implement the sample flow
algorithm given below. FSE implementations could easily be extended
to store, e.g., a flow's current sending rate for statistics
gathering or future potential optimizations.

5.3. Flows

Flows register themselves with SBD and FSE when they start,
deregister from the FSE when they stop, and carry out an UPDATE
function call every time their congestion controller calculates a new
sending rate. Via UPDATE, they provide the newly calculated rate and
optionally (if the algorithm supports it) the desired rate. The
desired rate is less than the calculated rate in case of application-
limited flows; otherwise, it is the same as the calculated rate.

Below, two example algorithms are described. While other algorithms
could be used instead, the same algorithm must be applied to all
flows. Names of variables used in the algorithms are explained
below.

o CC_R(f) - The rate received from the congestion controller of flow
   f when it calls UPDATE.

o FSE_R(f) - The rate calculated by the FSE for flow f.

o S_CR - The sum of the calculated rates of all flows in the same
   FG; this value is used to calculate the sending rate.

o FG - A group of flows having the same FGI, and hence sharing the
   same bottleneck.

o P(f) - The priority of flow f which is received from the flow's
   congestion controller; the FSE uses this variable for calculating
   FSE_R(f).

o S_P - The sum of all the priorities.

5.3.1. **Example algorithm 1 - Active FSE**

   This algorithm was designed to be the simplest possible method to
   assign rates according to the priorities of flows. Simulations
   results in [fse] indicate that it does however not significantly
   reduce queuing delay and packet loss.

   (1) When a flow f starts, it registers itself with SBD and the FSE.
        FSE_R(f) is initialized with the congestion controller's initial
        rate. SBD will assign the correct FGI. When a flow is assigned
        an FGI, it adds its FSE_R(f) to S_CR.

   (2) When a flow f stops or pauses, its entry is removed from the
        list.

   (3) Every time the congestion controller of the flow f determines a
        new sending rate CC_R(f), the flow calls UPDATE, which carries
        out the tasks listed below to derive the new sending rates for
        all the flows in the FG. A flow's UPDATE function uses a local
        (i.e. per-flow) temporary variable S_P, which is the sum of all
        the priorities.

        (a) It updates S_CR.

              S_CR = S_CR + CC_R(f) - FSE_R(f)

        (b) It calculates the sum of all the priorities, S_P.

              S_P = 0
              for all flows i in FG do
                  S_P = S_P + P(i)
              end for

        (c) It calculates the sending rates for all the flows in an FG
             and distributes them.

              for all flows i in FG do
                  FSE_R(i) = (P(i)*S_CR)/S_P
                  send FSE_R(i) to the flow i
              end for

5.3.2. **Example algorithm 2 - Conservative Active FSE**

   This algorithm extends algorithm 1 to conservatively emulate the
   behavior of a single flow by proportionally reducing the aggregate
   rate on congestion. Simulations results in [fse] indicate that it
   can significantly reduce queuing delay and packet loss.

   (1) When a flow f starts, it registers itself with SBD and the FSE.
        FSE_R(f) is initialized with the congestion controller's initial
        rate. SBD will assign the correct FGI. When a flow is assigned
        an FGI, it adds its FSE_R(f) to S_CR.

   (2) When a flow f stops or pauses, its entry is removed from the
        list.

   (3) Every time the congestion controller of the flow f determines a
        new sending rate CC_R(f), the flow calls UPDATE, which carries
        out the tasks listed below to derive the new sending rates for
        all the flows in the FG. A flow's UPDATE function uses a local
        (i.e. per-flow) temporary variable S_P, which is the sum of all
        the priorities, and a local variable DELTA, which is used to
        calculate the difference between CC_R(f) and the previously
        stored FSE_R(f). To prevent flows from either ignoring
        congestion or overreacting, a timer keeps them from changing
        their rates immediately after the common rate reduction that
        follows a congestion event. This timer is set to 2 RTTs of the
        flow that experienced congestion because it is assumed that a
        congestion event can persist for up to one RTT of that flow,
        with another RTT added to compensate for fluctuations in the
        measured RTT value.

        (a) It updates S_CR based on DELTA.

                if Timer has expired or not set then
                  DELTA = CC_R(f) - FSE_R(f)
                  if DELTA < 0 then // Reduce S_CR proportionally
                    S_CR = S_CR * CC_R(f) / FSE_R(f)
                    Set Timer for 2 RTTs
                  else
                    S_CR = S_CR + DELTA
                  end if
                end if

        (b) It calculates the sum of all the priorities, S_P.

```
            S_P = 0
            for all flows i in FG do
                S_P = S_P + P(i)
            end for
```

   (c) It calculates the sending rates for all the flows in an FG
        and distributes them.

```
            for all flows i in FG do
                FSE_R(i) = (P(i)*S_CR)/S_P
                send FSE_R(i) to the flow i
            end for
```

6. Application

   This section specifies how the FSE can be applied to specific
   congestion control mechanisms and makes general recommendations that
   facilitate applying the FSE to future congestion controls.

6.1. NADA

   Network-Assisted Dynamic Adapation (NADA) [I-D.ietf-rmcat-nada] is a
   congestion control scheme for rtcweb. It calculates a reference rate
   r_ref upon receiving an acknowledgment, and then, based on the
   reference rate, it calculates a video target rate r_vin and a sending
   rate for the flows, r_send.

   When applying the FSE to NADA, the UPDATE function call described in
   Section 5.3 gives the FSE NADA's reference rate r_ref. The
   recommended algorithm for NADA is the Active FSE in Section 5.3.1.
   In step 3 (c), when the FSE_R(i) is "sent" to the flow i, this means
   updating r_ref(r_vin and r_send) of flow i with the value of
   FSE_R(i).

6.2. General recommendations

   This section provides general advice for applying the FSE to
   congestion control mechanisms.

   Receiver-side calculations:
        When receiver-side calculations make assumptions about the rate
        of the sender, the calculations need to be synchronized or the
        receiver needs to be updated accordingly. This applies to TFRC
        [RFC5348], for example, where simulations showed somewhat less
        favorable results when using the FSE without a receiver-side
        change [fse].

Stateful algorithms:
> When a congestion control algorithm is stateful (e.g., TCP,
> with Slow Start, Congestion Avoidance and Fast Recovery), these
> states should be carefully considered such that the overall
> state of the aggregate flow is correct. This may require
> sharing more information in the UPDATE call.

Rate jumps:
> The FSE-based coupling algorithms can let a flow quickly
> increase its rate to its fair share, e.g. when a new flow joins
> or after a quiescent period. In case of window-based
> congestion controls, this may produce a burst which should be
> mitigated in some way. An example of how this could be done
> without using a timer is presented in [anrw2016], using TCP as
> an example.

7. Expected feedback from experiments

The algorithm described in this memo has so far been evaluated using
simulations covering all the tests for more than one flow from
[I-D.ietf-rmcat-eval-test] (see [IETF-93], [IETF-94]). Experiments
should confirm these results using at least the NADA congestion
control algorithm with real-life code (e.g., browsers communicating
over an emulated network covering the conditions in
[I-D.ietf-rmcat-eval-test]. The tests with real-life code should be
repeated afterwards in real network environments and monitored.
Experiments should investigate cases where the media coder's output
rate is below the rate that is calculated by the coupling algorithm
(FSE_R(i) in algorithms 1 and 2, section 5.3). Implementers and
testers are invited to document their findings in an Internet draft.

8. Acknowledgements

9. IANA Considerations

   This memo includes no request to IANA.

10. Security Considerations

   In scenarios where the architecture described in this document is
   applied across applications, various cheating possibilities arise:
   e.g., supporting wrong values for the calculated rate, the desired
   rate, or the priority of a flow. In the worst case, such cheating
   could either prevent other flows from sending or make them send at a
   rate that is unreasonably large. The end result would be unfair
   behavior at the network bottleneck, akin to what could be achieved
   with any UDP based application. Hence, since this is no worse than
   UDP in general, there seems to be no significant harm in using this
   in the absence of UDP rate limiters.

   In the case of a single-user system, it should also be in the
   interest of any application programmer to give the user the best
   possible experience by using reasonable flow priorities or even
   letting the user choose them. In a multi-user system, this interest
   may not be given, and one could imagine the worst case of an "arms
   race" situation, where applications end up setting their priorities
   to the maximum value. If all applications do this, the end result is
   a fair allocation in which the priority mechanism is implicitly
   eliminated, and no major harm is done.

11. References

11.1. Normative References

   [I-D.ietf-rmcat-nada]
              Zhu, X., Pan, R., Ramalho, M., Cruz, S., Jones, P., Fu,
              J., D'Aronco, S., and C. Ganzhorn, "NADA: A Unified
              Congestion Control Scheme for Real-Time Media", draft-
              ietf-rmcat-nada-03 (work in progress), September 2016.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
              RFC2119, March 1997,
              <http://www.rfc-editor.org/info/rfc2119>.

   [RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager",
              RFC 3124, DOI 10.17487/RFC3124, June 2001,
              <http://www.rfc-editor.org/info/rfc3124>.

   [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP
              Friendly Rate Control (TFRC): Protocol Specification", RFC
              5348, DOI 10.17487/RFC5348, September 2008,
              <http://www.rfc-editor.org/info/rfc5348>.

11.2. Informative References

   [anrw2016]
              Islam, S. and M. Welzl, "Start Me Up:Determining and
              Sharing TCP's Initial Congestion Window", ACM, IRTF, ISOC
              Applied Networking Research Workshop 2016 (ANRW 2016) ,
              2016.

   [fse]   Islam, S., Welzl, M., Gjessing, S., and N. Khademi,
              "Coupled Congestion Control for RTP Media", ACM SIGCOMM
              Capacity Sharing Workshop (CSWS 2014) and ACM SIGCOMM CCR
              44(4) 2014; extended version available as a technical
              report from
              http://safiquli.at.ifi.uio.no/paper/fse-tech-report.pdf ,
              2014.

   [fse-noms]
              Islam, S., Welzl, M., Hayes, D., and S. Gjessing,
              "Managing Real-Time Media Flows through a Flow State
              Exchange", IEEE NOMS 2016, Istanbul, Turkey , 2016.

   [I-D.ietf-rmcat-eval-test]
              Sarker, Z., Singh, V., Zhu, X., and M. Ramalho, "Test
              Cases for Evaluating RMCAT Proposals", draft-ietf-rmcat-
              eval-test-04 (work in progress), October 2016.

   [I-D.ietf-rmcat-gcc]
              Holmer, S., Lundin, H., Carlucci, G., Cicco, L., and S.
              Mascolo, "A Google Congestion Control Algorithm for Real-
              Time Communication", draft-ietf-rmcat-gcc-02 (work in
              progress), July 2016.

   [I-D.ietf-rmcat-sbd]
              Hayes, D., Ferlin, S., Welzl, M., and K. Hiorth, "Shared
              Bottleneck Detection for Coupled Congestion Control for
              RTP Media.", draft-ietf-rmcat-sbd-04 (work in progress),
              March 2016.

   [I-D.ietf-rtcweb-transports]
              Alvestrand, H., "Transports for WebRTC", Internet-draft
              draft-ietf-rtcweb-transports-17.txt, October 2016.

   [IETF-93] Islam, S., Welzl, M., and S. Gjessing, "Updates on Coupled
              Congestion Control for RTP Media", July 2015,
              <https://www.ietf.org/proceedings/93/rmcat.html>.

   [IETF-94] Islam, S., Welzl, M., and S. Gjessing, "Updates on Coupled
              Congestion Control for RTP Media", November 2015,
              <https://www.ietf.org/proceedings/94/rmcat.html>.

   [RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-
              Time Communication Use Cases and Requirements", RFC 7478,
              DOI 10.17487/RFC7478, March 2015,
              <http://www.rfc-editor.org/info/rfc7478>.

   [RFC7656] Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and
              B. Burman, Ed., "A Taxonomy of Semantics and Mechanisms
              for Real-Time Transport Protocol (RTP) Sources", RFC 7656,
              DOI 10.17487/RFC7656, November 2015,
              <http://www.rfc-editor.org/info/rfc7656>.

   [rtcweb-rtp-usage]
              Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time
              Communication (WebRTC): Media Transport and Use of RTP",
              Internet-draft draft-ietf-rtcweb-rtp-usage-26.txt, March
              2016.

   [transport-multiplex]
              Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a
              Single Lower-Layer Transport", Internet-draft draft-
              westerlund-avtcore-transport-multiplexing-07.txt, October
              2013.

Appendix A. Application to GCC

   Google Congestion Control (GCC) [I-D.ietf-rmcat-gcc] is another
   congestion control scheme for RTP flows that is under development.
   GCC is not yet finalised, but at the time of this writing, the rate
   control of GCC employs two parts: controlling the bandwidth estimate
   based on delay, and controlling the bandwidth estimate based on loss.
   Both are designed to estimate the available bandwidth, A_hat.

   When applying the FSE to GCC, the UPDATE function call described in
   Section 5.3 gives the FSE GCC's estimate of available bandwidth
   A_hat. The recommended algorithm for GCC is the Active FSE in
   Section 5.3.1. In step 3 (c), when the FSE_R(i) is "sent" to the
   flow i, this means updating A_hat of flow i with the value of
   FSE_R(i).

Appendix B. **Scheduling**

   When connections originate from the same host, it would be possible
   to use only one single sender-side congestion controller which
   determines the overall allowed sending rate, and then use a local
   scheduler to assign a proportion of this rate to each RTP session.
   This way, priorities could also be implemented as a function of the
   scheduler. The Congestion Manager (CM) [RFC3124] also uses such a
   scheduling function.

Appendix C. **Example algorithm - Passive FSE**

   Active algorithms calculate the rates for all the flows in the FG and
   actively distribute them. In a passive algorithm, UPDATE returns a
   rate that should be used instead of the rate that the congestion
   controller has determined. This can make a passive algorithm easier
   to implement; however, when round-trip times of flows are unequal,
   shorter-RTT flows may (depending on the congestion control algorithm)
   update and react to the overall FSE state more often than longer-RTT
   flows, which can produce unwanted side effects. This problem is more
   significant when the congestion control convergence depends on the
   RTT. While the passive algorithm works better for congestion
   controls with RTT-independent convergence, it can still produce
   oscillations on short time scales. The algorithm described below is
   therefore considered as highly experimental and not safe to deploy
   outside of testbed environments. Results of a simplified passive FSE
   algorithm with both NADA and GCC can be found in [fse-noms].

   This passive version of the FSE stores the following information in
   addition to the variables described in Section 5.2:

   o The desired rate DR(f) of flow f. This can be smaller than the
      calculated rate if the application feeding into the flow has less
      data to send than the congestion controller would allow. In case
      of a bulk transfer, DR(f) must be set to CC_R(f) received from the
      congestion module of flow f.

   The passive version of the FSE contains one static variable per FG
   called TLO (Total Leftover Rate -- used to let a flow 'take'
   bandwidth from application-limited or terminated flows) which is
   initialized to 0. For the passive version, S_CR is limited to
   increase or decrease as conservatively as a flow's congestion
   controller decides in order to prohibit sudden rate jumps.

   (1) When a flow f starts, it registers itself with SBD and the FSE.
        FSE_R(f) and DR(f) are initialized with the congestion
        controller's initial rate. SBD will assign the correct FGI.
        When a flow is assigned an FGI, it adds its FSE_R(f) to S_CR.

(2) When a flow f stops or pauses, it sets its DR(f) to 0 and sets
     P(f) to -1.

(3) Every time the congestion controller of the flow f determines a
     new sending rate CC_R(f), assuming the flow's new desired rate
     new_DR(f) to be "infinity" in case of a bulk data transfer with
     an unknown maximum rate, the flow calls UPDATE, which carries
     out the tasks listed below to derive the flow's new sending
     rate, Rate(f). A flow's UPDATE function uses a few local (i.e.
     per-flow) temporary variables, which are all initialized to 0:
     DELTA, new_S_CR and S_P.

     (a) For all the flows in its FG (including itself), it
          calculates the sum of all the calculated rates, new_S_CR.
          Then it calculates DELTA: the difference between FSE_R(f)
          and CC_R(f).

```
             for all flows i in FG do
                 new_S_CR = new_S_CR + FSE_R(i)
             end for
             DELTA =  CC_R(f) - FSE_R(f)
```

     (b) It updates S_CR, FSE_R(f) and DR(f).

```
             FSE_R(f) = CC_R(f)
             if DELTA > 0 then  // the flow's rate has increased
                 S_CR = S_CR + DELTA
             else if DELTA < 0 then
                 S_CR = new_S_CR + DELTA
             end if
             DR(f) = min(new_DR(f),FSE_R(f))
```

     (c) It calculates the leftover rate TLO, removes the terminated
          flows from the FSE and calculates the sum of all the
          priorities, S_P.

```
            for all flows i in FG do
               if P(i)<0 then
                  delete flow
               else
                  S_P = S_P + P(i)
               end if
            end for
            if DR(f) < FSE_R(f) then
               TLO = TLO + (P(f)/S_P) * S_CR - DR(f))
            end if
```

(d) It calculates the sending rate, Rate(f).

```
         Rate(f) = min(new_DR(f), (P(f)*S_CR)/S_P + TLO)

         if Rate(f) != new_DR(f) and TLO > 0 then
             TLO = 0  // f has 'taken' TLO
         end if
```

(e) It updates DR(f) and FSE_R(f) with Rate(f).

```
         if Rate(f) > DR(f) then
             DR(f) = Rate(f)
         end if
         FSE_R(f)  = Rate(f)
```

The goals of the flow algorithm are to achieve prioritization,
improve network utilization in the face of application-limited flows,
and impose limits on the increase behavior such that the negative
impact of multiple flows trying to increase their rate together is
minimized. It does that by assigning a flow a sending rate that may
not be what the flow's congestion controller expected. It therefore
builds on the assumption that no significant inefficiencies arise
from temporary application-limited behavior or from quickly jumping
to a rate that is higher than the congestion controller intended.
How problematic these issues really are depends on the controllers in
use and requires careful per-controller experimentation. The coupled
congestion control mechanism described here also does not require all
controllers to be equal; effects of heterogeneous controllers, or
homogeneous controllers being in different states, are also subject
to experimentation.

This algorithm gives all the leftover rate of application-limited
flows to the first flow that updates its sending rate, provided that
this flow needs it all (otherwise, its own leftover rate can be taken
by the next flow that updates its rate). Other policies could be

applied, e.g. to divide the leftover rate of a flow equally among all
other flows in the FGI.

C.1. Example operation (passive)

In order to illustrate the operation of the passive coupled
congestion control algorithm, this section presents a toy example of
two flows that use it. Let us assume that both flows traverse a
common 10 Mbit/s bottleneck and use a simplistic congestion
controller that starts out with 1 Mbit/s, increases its rate by 1
Mbit/s in the absence of congestion and decreases it by 2 Mbit/s in
the presence of congestion. For simplicity, flows are assumed to
always operate in a round-robin fashion. Rate numbers below without
units are assumed to be in Mbit/s. For illustration purposes, the
actual sending rate is also shown for every flow in FSE diagrams even
though it is not really stored in the FSE.

Flow #1 begins. It is a bulk data transfer and considers itself to
have top priority. This is the FSE after the flow algorithm's step
1:

```
-------------------------------------------
| # | FGI |  P  | FSE_R  |  DR  | Rate |
|   |     |     |        |      |      |
| 1 |  1  |  1  |   1    |  1   |  1   |
-------------------------------------------
S_CR = 1, TLO = 0
```

Its congestion controller gradually increases its rate. Eventually,
at some point, the FSE should look like this:

```
-------------------------------------------
| # | FGI |  P  |  FSE_R  |  DR  | Rate |
|   |     |     |         |      |      |
| 1 |  1  |  1  |   10    |  10  |  10  |
-------------------------------------------
S_CR = 10, TLO = 0
```

Now another flow joins. It is also a bulk data transfer, and has a
lower priority (0.5):

```
---------------------------------------------
| # | FGI |   P   | FSE_R  |  DR  | Rate |
|   |     |       |        |      |      |
| 1 |  1  |   1   |   10   |  10  |  10  |
| 2 |  1  |  0.5  |    1   |   1  |   1  |
---------------------------------------------
S_CR = 11, TLO = 0
```

Now assume that the first flow updates its rate to 8, because the
total sending rate of 11 exceeds the total capacity. Let us take a
closer look at what happens in step 3 of the flow algorithm.

CC_R(1) = 8. new_DR(1) = infinity.
3 a) new_S_CR = 11; DELTA = 8 - 10 = -2.
3 b) FSE_R(1) = 8. DELTA is negative, hence S_CR = 9;
     DR(1) = 8.
3 c) S_P = 1.5.
3 d) new sending rate Rate(1) = min(infinity, 1/1.5 * 9 + 0) = 6.
3 e) FSE_R(1) = 6.

The resulting FSE looks as follows:
```
---------------------------------------------
| # | FGI |   P   | FSE_R  |  DR  | Rate |
|   |     |       |        |      |      |
| 1 |  1  |   1   |    6   |   8  |   6  |
| 2 |  1  |  0.5  |    1   |   1  |   1  |
---------------------------------------------
S_CR = 9, TLO = 0
```

The effect is that flow #1 is sending with 6 Mbit/s instead of the 8
Mbit/s that the congestion controller derived. Let us now assume
that flow #2 updates its rate. Its congestion controller detects
that the network is not fully saturated (the actual total sending
rate is 6+1=7) and increases its rate.

```
CC_R(2) = 2. new_DR(2) = infinity.
3 a) new_S_CR = 7; DELTA = 2 - 1 = 1.
3 b) FSE_R(2) = 2. DELTA is positive, hence S_CR = 9 + 1 = 10;
     DR(2) = 2.
3 c) S_P = 1.5.
3 d) Rate(2) = min(infinity, 0.5/1.5 * 10 + 0) = 3.33.
3 e) DR(2) = FSE_R(2) = 3.33.
```

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
1	1	1	6	8	6
2	1	0.5	3.33	3.33	3.33

S_CR = 10, TLO = 0

The effect is that flow #2 is now sending with 3.33 Mbit/s, which is
close to half of the rate of flow #1 and leads to a total utilization
of 6(#1) + 3.33(#2) = 9.33 Mbit/s. Flow #2's congestion controller
has increased its rate faster than the controller actually expected.
Now, flow #1 updates its rate. Its congestion controller detects
that the network is not fully saturated and increases its rate.
Additionally, the application feeding into flow #1 limits the flow's
sending rate to at most 2 Mbit/s.

```
CC_R(1) = 7. new_DR(1) = 2.
3 a) new_S_CR = 9.33; DELTA = 1.
3 b) FSE_R(1) = 7, DELTA is positive, hence S_CR = 10 + 1 = 11;
     DR(1) = min(2, 7) = 2.
3 c) S_P = 1.5; DR(1) < FSE_R(1), hence TLO = 1/1.5 * 11 - 2 = 5.33.
3 d) Rate(1) = min(2, 1/1.5 * 11 + 5.33) = 2.
3 e) FSE_R(1) = 2.
```

The resulting FSE looks as follows:

#	FGI	P	FSE_R	DR	Rate
1	1	1	2	2	2
2	1	0.5	3.33	3.33	3.33

S_CR = 11, TLO = 5.33

Now, the total rate of the two flows is 2 + 3.33 = 5.33 Mbit/s, i.e.
the network is significantly underutilized due to the limitation of
flow #1. Flow #2 updates its rate. Its congestion controller
detects that the network is not fully saturated and increases its
rate.

CC_R(2) = 4.33. new_DR(2) = infinity.
3 a) new_S_CR = 5.33; DELTA = 1.
3 b) FSE_R(2) = 4.33. DELTA is positive, hence S_CR = 12;
     DR(2) = 4.33.
3 c) S_P = 1.5.
3 d) Rate(2) = min(infinity, 0.5/1.5 * 12 + 5.33) = 9.33.
3 e) FSE_R(2) = 9.33, DR(2) = 9.33.

The resulting FSE looks as follows:
```
----------------------------------------------
| # | FGI |   P   |  FSE_R  |  DR   | Rate |
|   |     |       |         |       |      |
| 1 |  1  |   1   |    2    |   2   |   2  |
| 2 |  1  |  0.5  |   9.33  | 9.33  | 9.33 |
----------------------------------------------
S_CR = 12, TLO = 0
```


Now, the total rate of the two flows is 2 + 9.33 = 11.33 Mbit/s.
Finally, flow #1 terminates. It sets P(1) to -1 and DR(1) to 0. Let
us assume that it terminated late enough for flow #2 to still
experience the network in a congested state, i.e. flow #2 decreases
its rate in the next iteration.

```
CC_R(2) = 7.33. new_DR(2) = infinity.
3 a) new_S_CR = 11.33; DELTA = -2.
3 b) FSE_R(2) = 7.33. DELTA is negative, hence S_CR = 9.33;
     DR(2) = 7.33.
3 c) Flow 1 has P(1) = -1, hence it is deleted from the FSE.
     S_P = 0.5.
3 d) Rate(2) = min(infinity, 0.5/0.5*9.33 + 0) = 9.33.
3 e) FSE_R(2) = DR(2) = 9.33.
```

The resulting FSE looks as follows:
```
---------------------------------------------
| # | FGI |  P    | FSE_R  |  DR  | Rate |
|   |     |       |        |      |      |
| 2 |  1  |  0.5  |  9.33  | 9.33 | 9.33 |
---------------------------------------------
S_CR = 9.33, TLO = 0
```

Appendix D. Change log

D.1. draft-welzl-rmcat-coupled-cc

D.1.1. Changes from -00 to -01

o Added change log.

o Updated the example algorithm and its operation.

D.1.2. Changes from -01 to -02

o Included an active version of the algorithm which is simpler.

o Replaced "greedy flow" with "bulk data transfer" and "non-greedy"
   with "application-limited".

o Updated new_CR to CC_R, and CR to FSE_R for better understanding.

D.1.3. Changes from -02 to -03

o Included an active conservative version of the algorithm which
   reduces queue growth and packet loss; added a reference to a
   technical report that shows these benefits with simulations.

o Moved the passive variant of the algorithm to appendix.

D.1.4. **Changes from -03 to -04**

   o Extended SBD section.

   o Added a note about window-based controllers.

D.1.5. **Changes from -04 to -05**

   o Added a section about applying the FSE to specific congestion
      control algorithms, with a subsection specifying its use with
      NADA.

D.2. **draft-ietf-rmcat-coupled-cc**

D.2.1. **Changes from draft-welzl-rmcat-coupled-cc-05**

   o Moved scheduling section to the appendix.

D.2.2. **Changes from -00 to -01**

   o Included how to apply the algorithm to GCC.

   o Updated variable names of NADA to be in line with the latest
      version.

   o Added a reference to [I-D.ietf-rtcweb-transports] to make a
      connection to the prioritization text there.

D.2.3. **Changes from -01 to -02**

   o Minor changes.

   o Moved references of NADA and GCC from informative to normative.

   o Added a reference for the passive variant of the algorithm.

D.2.4. **Changes from -02 to -03**

   o Minor changes.

   o Added a section about expected feedback from experiments.

D.2.5. **Changes from -03 to -04**

   o Described the names of variables used in the algorithms.

   o Added a diagram to illustrate the interaction between flows and
      the FSE.

   o Added text on the trade-off of using the configuration based
      approach.

   o Minor changes to enhance the readability.

[D.2.6](#). Changes from -04 to -05

   o Changed several occurrences of "NADA and GCC" to "NADA", including
      the abstract.

   o Moved the application to GCC to an appendix, and made the GCC
      reference informative.

   o Provided a few more general recommendations on applying the
      coupling algorithm.

[D.2.7](#). Changes from -05 to -06

   o Incorporated comments by Colin Perkins.

Authors' Addresses

   Safiqul Islam
   University of Oslo
   PO Box 1080 Blindern
   Oslo N-0316
   Norway

   Phone: +47 22 84 08 37
   Email: safiquli@ifi.uio.no

   Michael Welzl
   University of Oslo
   PO Box 1080 Blindern
   Oslo N-0316
   Norway

   Phone: +47 22 85 24 20
   Email: michawe@ifi.uio.no

Stein Gjessing
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Phone: +47 22 85 24 44
Email: steing@ifi.uio.no