

TSVWG  
Internet-Draft  
Intended status: Informational  
Expires: September 4, 2014

K. Nichols  
Pollere, Inc.  
V. Jacobson  
Google  
March 3, 2014

Controlled Delay Active Queue Management  
draft-nichols-tsvwg-codel-02

Abstract

The "persistently full buffer" problem has been discussed in the IETF community since the early 80's [RFC896]. The IRTF's End-to-End Working Group called for the deployment of active queue management (AQM) to solve the problem in 1998 [RFC2309]. Despite the awareness, the problem gotten worse as by Moore's Law growth in memory density fueled an exponential increase in buffer pool size. Efforts to deploy AQM have been frustrated by difficult configuration and negative impact on network utilization. The full buffer problem, recently christened "bufferbloat" [TSVBB2011, BB2011] has become increasingly important throughout the Internet but particularly at the consumer edge.

To address bufferbloat, this document describes a general framework for controlling excessived delay in networks called Controlled Delay (CoDel) designed to work in modern networking environments as a part of the solution to bufferbloat [CODEL2012]. CoDel consists of an estimator, a setpoint, and a control loop and can be deployed in the Internet without configuration. CoDel comprises some major technical innovations and has been made available as open source so that the framework can be applied by the community to a range of problems. It has been implemented in Linux (and available in the Linux distribution) and deployed in some networks at the consumer edge. In addition, the framework has been successfully applied in other ways.

Note: Code Components extracted from this document must include the license as included with the code in Section 5.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 4, 2014.

#### Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### 1. Introduction

The need for queue management has been evident for decades. Recently the need has become more critical due to the increased consumer use of the Internet mixing large video transactions with time-critical VoIP and gaming. Gettys [TSV2011, BB2011] has been instrumental in publicizing the problem and the measurement work [CHARB2007, NATAL2010] and coining the term bufferbloat. Large content distributors such as Google have observed that bufferbloat is ubiquitous and adversely effects performance for many users. The solution is an effective AQM that remediates bufferbloat at a bottleneck while "doing no harm" at hops where buffers are not bloated.

The development and deployment of effective active queue management has been hampered by persistent misconceptions about the cause and meaning of queues. Network buffers exist to absorb the packet bursts that occur naturally in statistically multiplexed networks. Short-term mismatches in traffic arrival and departure rates that arise from upstream resource contention, transport conversation startup transients and/or changes in the number of conversations sharing a link create queues. Unfortunately, other network behavior can cause queues to fill and their effects aren't nearly as benign. Discussion of these issues and why the solution isn't just smaller buffers can be found in [RFC2309], [VANQ2006], [REDL1998] and [CODEL2012]. It is

critical to understand the difference between the necessary, useful "good" queue and the counterproductive "bad" queue.

Many approaches to active queue management (AQM) have been developed over the past two decades but none has been widely deployed due to performance problems. When designed with the wrong conceptual model for queues, AQMs have limited operational range, require a lot of configuration tweaking, and frequently impair rather than improve performance. Today, the demands on an effective AQM are even greater: many network devices must work across a range of bandwidths, either due to link variations or due to the mobility of the device. The CoDel approach is designed to meet the following goals:

- o is parameterless for normal operation - has no knobs for operators, users, or implementers to adjust
- o treats "good queue" and "bad queue" differently, that is, keeps delay low while permitting necessary bursts of traffic
- o controls delay while insensitive (or nearly so) to round trip delays, link rates and traffic loads; this goal is to "do no harm" to network traffic while controlling delay
- o adapts to dynamically changing link rates with no negative impact on utilization
- o is simple and efficient (can easily span the spectrum from low-end, linux-based access points and home routers up to high-end commercial router silicon)

Since April, 2012, when CoDel was published, a number of talented and enthusiastic implementers have been using and adapting it with promising results. Much of this work is collected at: <http://www.bufferbloat.net/projects/codel>. CoDel has five major innovations that distinguish it from prior AQMs: use of local queue minimum to track congestion ("bad queue"), use of an efficient single state variable representation of that tracked statistic, use of packet sojourn time as the observed datum, rather than packets, bytes, or rates, use of mathematically determined setpoint derived from maximizing the network power metric, and a modern state space controller.

CoDel configures itself based on a round-trip time metric which can be set to 100ms for the normal, terrestrial Internet. With no changes to parameters, we have found CoDel to work across a wide range of conditions, with varying links and the full range of terrestrial round trip times. CoDel has been implemented in Linux very efficiently and should lend itself to silicon implementation.

CoDel is well-adapted for use in multiple queued devices and has been used by Eric Dumazet with multiple queues in sophisticated queue management approach, `fq_codel` (covered in another draft).

## 2. Conventions used in this document

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 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

## 3. Building Blocks of Queue Management

Two decades of work on queue management failed to yield an approach that could be widely deployed in the Internet. With careful tuning for particular usages, queue management techniques have been able to "kind of" work, that is decrease queuing delays, but utilization and fairness suffer unduly. At the heart of queue management is the notion of "good queue" and "bad queue" and the search for ways to get rid of the bad queue (which only adds delay) while preserving the good queue (which provides for good utilization). This section explains queuing, both good and bad, and covers the innovative CoDel building blocks that can be used to manage packet buffers to keep their queues in the "good" range.

Packet queues form in buffers facing bottleneck links, i.e., where the line rate goes from high to low or many links converge. The well-known bandwidth-delay product (sometimes called "pipe size") is the bottleneck's bandwidth multiplied by the sender-receiver-sender round-trip delay and is the amount of data that has to be in transit between two hosts in order to run at 100% utilization. To explore how queues can form, consider a long-lived TCP connection with a 25 packet window sending through a connection with a bandwidth-delay product of 20 packets. After an initial burst of packets the connection will settle into a five packet (+/-1) standing queue, the size determined by the window mismatch to the pipe size and unrelated to the connection's sending rate. The connection has 25 packets in flight at all times, but only 20 packets arrive at the destination over a round trip time. If the TCP connection has a 30 packet window, the queue will be ten packets with no change in sending rate.

Similarly, if the window is 20 packets, there will be no queue but the sending rate is the same. Nothing can be inferred about the sender rate from the queue and the existence of any queue at all other than transient bursts can only create delay in the network. The sender needs to reduce the number of packets in flight rather than sending rate.

In the above example, the five packet standing queue can be seen to contribute nothing but delay to the connection thus is clearly "bad queue". If, in our example, there is a single bottleneck link and it is much slower than the link that feeds it (say, a high-speed ethernet link into a limited DSL uplink) a 20 packet buffer at the bottleneck might be necessary to temporarily hold the 20 packets in flight to keep the utilization high. The burst of packets should drain completely (to 0 or 1 packets) within a round trip time and this transient queue is "good queue" because it allows the connection to keep the 20 packets in flight and for the bottleneck link to be fully utilized. In terms of the delay experienced we can observe that "good queue" goes away in about a round trip time, while "bad queue" hangs around causing delays.

Effective queue management detects "bad queue" while ignoring "good queue" and takes action to get rid of the bad queue when it is detected. The goal is a queue controller that accomplishes this objective. To control queue, we need three basic components

- o Estimator - figure out what we've got
- o Setpoint - know what we want
- o Control loop - if what we've got isn't what we want, we need a way to move it there

### 3.1. Estimator

The Estimator both observes the queue and detects when good queue turns to bad queue and vice versa. CoDel has two innovations in its Estimator: what is observed as an indicator of queue and how the observations are used to detect good/bad queue.

In the past, queue length has been widely used as an observed indicator of congestion and is frequently conflated with sending rate. Use of queue length as a metric is sensitive to how and when the length is observed. A high speed arrival link to a buffer serviced at a much lower rate can rapidly build up a queue that might disperse completely or down to a single packet before a round trip time has elapsed. If the queue length is monitored at packet arrival (as in original RED) or departure time, every packet will see a queue

with one possible exception. If the queue length itself is time sampled (as recommended in [REDL1998], a truer picture of the queue's occupancy can be gained but a separate process is required.

The use of queue length is further complicated in networks that are subject to both short and long term changes in available link rate (as in wifi). Link rate drops can result in a spike in queue length that should be ignored unless it persists. The length metric is problematic when what we really want to control is the amount of excess delay packets experience due to a persistent or standing queue. The sojourn time that a packet spends in the buffer is exactly what we want to track. Tracking the packet sojourn times in the buffer observes the actual delay experienced by each packet. Sojourn time is independent of link rate, gives superior performance to use of buffer size, and is directly related to the user-visible performance. It works regardless of line rate changes or whether the link is shared by multiple queues (which the individual queues may experience as changing rates).

Consider a link shared by two queues, one priority queue and one of lower priority. Packets that arrive to the high priority queue are sent as soon as the link is available while packets of the other queue have to wait till the the priority queue is empty (i.e., a strict priority scheduler). The number of packets in the priority queue might be large but the queue is emptied quickly and the amount of time each packet spends enqueued (the sojourn time) is not large. The other queue might have a smaller number of packets, but packet sojourn times will include the wait for the high priority packets to be sent. This makes the sojourn times a good sample of the congestion that each separate queue is experiencing and shows how this metric is independent of the number of queues used or the service discipline and instead reflective of the congestion seen by the individual queue.

With sojourn time as the observation, how can it be used to separate good queue from bad queue? In the past, averages, in particular of queue length, have been used to determine bad queue. Consider the burst that disperses every round trip time. The average queue will be one-half the burst size, though this might vary depending on when the average is computed and the timing of arrivals. The average then would indicate a persistent queue where there is none. If instead we track the minimum observation, if there is one packet that has a zero sojourn time then there is no persistent queue. The value of the minimum in detecting persistent queue is apparent when looking at graphs of queue delay.

The standing queue can be detected by tracking the (local) minimum queue delay packets experience. To ensure that this minimum value

does not become stale, it has to have been experienced recently, i.e. during an appropriate past time interval. This "interval" is the maximum amount of time a minimum is considered to be in effect. It is clear that this interval should be at least a round trip time to avoid falsely detecting a persistent queue and not a lot more than a round trip time to avoid delay in detecting the persistent queue. This suggests that the appropriate interval value is the maximum round-trip time of all the connections sharing the buffer. To avoid outlier values, the 95-99th percentile value is preferred rather than a strict maximum.

A key realization makes the local minimum an efficiently computed statistic. Note that it is sufficient to keep a single state variable of how long the minimum has been above or below a target value rather than retaining all the local values to compute the minimum, leading to both storage and computational savings.

These two innovations, use of sojourn time as observed values and the local minimum as the statistic to monitor queue congestion are key to CoDel's Estimator building block. The local minimum sojourn time provides an accurate and robust measure of standing queue and has an efficient implementation. In addition, use of the minimum sojourn time has important advantages in implementation. The minimum packet sojourn can only be decreased when a packet is dequeued which means that all the work of CoDel can take place when packets are dequeued for transmission and that no locks are needed in the implementation. The minimum is the only statistic with this property.

A more detailed explanation with many pictures can be found at: <http://pollere.net/Pdfdocs/GrantJul06.pdf> and <http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf>.

### 3.2. Setpoint

Now that we have a robust way of detecting standing queue, we need to have a Setpoint that tells us when to act. If the controller is set to take action as soon as the estimator has a non-zero value, the average drop rate will be maximized which minimizes TCP goodput [MACTCP1997]. Also, since this policy results in no backlog over time (no persistent queue), it also maximizes the bottleneck link bandwidth lost because of normal stochastic variation in packet interarrival time and obliterates much of the value of having a buffer. We want a setpoint that maximizes utilization while minimizing delay. Early in the history of packet networking, Kleinrock developed the analytic machinery to do this using a quantity he called `'power'` (the ratio of a normalized throughput to a normalized delay) [KLEIN81].

It's straightforward to derive an analytic expression the average goodput of a TCP conversation for a given round-trip time  $\_r\_$  and setpoint  $\_f\_$  (where  $\_f\_$  is expressed as a fraction of  $\_r\_$ ) [VJTARG14]. Reno TCP, for example, yields:

$$\text{goodput} = \_r\_ (3 + 6\_f\_ - \_f\_^2) / (4 (1 + \_f\_))$$

Since the peak delay is just  $\_f\_ r\_$ , it's clear that  $\_power\_$  is solely a function of  $\_f\_$  since the  $\_r\_$ 's in the numerator and denominator cancel:

As Kleinrock observed, the best operating point, in terms of bandwidth / delay tradeoff, is the peak power point since points off the peak represent a higher cost (in delay) per unit of bandwidth. The power vs.  $\_f\_$  curve for any AIMD TCP is monotone decreasing. But the curve is very flat for  $\_f\_ < 0.1$  followed by a increasing curvature with a knee around .2 then a steep, almost linear fall off [TSV84] [VJTARG14]. Since the previous equation showed that goodput is monotone increasing with  $\_f\_$ , the best operating point is near the right edge of the flat top since that represents the highest goodput achievable for a negligible increase in delay. However, since the  $\_r\_$  in the model is a conservative upper bound, a target of  $.1\_r\_$  runs the risk of pushing shorter RTT connections over the knee and giving them higher delay for no significant goodput increase. Generally, a more conservative target of  $.05\_r\_$  offers a good utilization vs. delay tradeoff while giving enough headroom to work well with a large variation in real RTT.

As the above analysis shows, a very small standing queue gives close to 100% utilization. While this result was for Reno TCP, the derivation uses only properties that must hold for any 'TCP friendly' transport. We have verified by both analysis and simulation that this result holds for Reno, Cubic, and Westwood [TSV84]. This results in a particularly simple form for the setpoint: the ideal range for the permitted standing queue is between 5 and 10% of the TCP connection RTT. Thus  $\_target\_$  is simply 5% of the  $\_interval\_$  of section 3.1.

### 3.3. Control Loop

Section 3.1 describes a simple, reliable way to measure bad (persistent) queue. Section 3.2 shows that TCP congestion control dynamics gives rise to a setpoint for this measure that's a provably good balance between enhancing throughput and minimizing delay, and that this setpoint is a constant fraction of the same 'largest average RTT' interval used to distinguish persistent from transient queue. The only remaining building block needed for a basic AQM is a 'control loop' algorithm to effectively drive the queuing system from



any 'persistent queue above target' state to a state where the persistent queue is below target.

Control theory provides a wealth of approaches to the design of control loops. Most of classical control theory deals with the control of linear, time-invariant, single-input-single-output (SISO) systems. Control loops for these systems generally come from a (well understood) class known as Proportional-Integral-Derivative (PID) controllers. Unfortunately, a queue is not a linear system and an AQM operates at the point of maximum non-linearity (where the output link bandwidth saturates so increased demand creates delay rather than higher utilization). Output queues are also not time-invariant since traffic is generally a mix of connections which start and stop at arbitrary times and which can have radically different behaviors ranging from "open loop" UDP audio/video to "closed-loop" congestion-avoiding TCP. Finally, the constantly changing mix of connections (which can't be converted to a single 'lumped parameter' model because of their transfer function differences) makes the system multi-input-multi-output (MIMO), not SISO.

Since queuing systems match none of the prerequisites for a classical controller, a modern state-space controller is a better approach with states 'no persistent queue' and 'has persistent queue'. Since Internet traffic mixtures change rapidly and unpredictably, a noise and error tolerant adaptation algorithm like Stochastic Gradient is a good choice. Since there's essentially no information in the amount of persistent queue [TSV84], the adaptation should be driven by how long it has persisted.

Consider the two extremes of traffic behavior, a single open-loop UDP video stream and a single, long-lived TCP bulk data transfer. If the average bandwidth of the UDP video stream is greater than the bottleneck link rate, the link's queue will grow and the controller will eventually enter 'has persistent queue' state and start dropping packets. Since the video stream is open loop, its arrival rate is unaffected by drops so the queue will persist until the average drop rate is greater than the output bandwidth deficit ( $= \text{average arrival rate} - \text{average departure rate}$ ) so the job of the adaptation algorithm is to discover this rate. For this example, the adaptation could consist of simply estimating the arrival and departure rates then dropping at a rate slightly greater than their difference. But this class of algorithm won't work at all for the bulk data TCP stream. TCP runs in closed-loop flow balance [TSV84] so its arrival rate is almost always exactly equal to the departure rate - the queue isn't the result of a rate imbalance but rather a mismatch between the TCP sender's window and the src-dst-src round-trip path capacity (i.e., the connection's bandwidth\*delay product). The sender's TCP congestion avoidance algorithm will slowly increase the send window

(one packet per round-trip-time) [RFC2581] which will eventually cause the bottleneck to enter 'has persistent queue' state. But, since the average input rate is the same as the average output rate, the rate deficit estimation that gave the correct drop rate for the video stream would compute a drop rate of zero for the TCP stream. However, if the output link drops one packet as it enters 'has persistent queue' state, when the sender discovers this (via TCP's normal packet loss repair mechanisms) it will reduce its window by a factor of two [RFC2581] so, one round-trip-time after the drop, the persistent queue will go away.

If there were  $N$  TCP conversations sharing the bottleneck, the controller would have to drop  $O(N)$  packets, one from each conversation, to make all the conversations reduce their window to get rid of the persistent queue. If the traffic mix consists of short ( $\leq \text{bandwidth} \times \text{delay}$  product) conversations, the aggregate behavior becomes more like the open-loop video example since each conversation is likely to have already sent all its packets by the time it learns about a drop so each drop has negligible effect on subsequent traffic.

The controller doesn't know what type, how many or how long are the conversations creating its queue so it has to learn that. Since single drops can have a large effect if the degree of multiplexing (the number of active conversations) is small, dropping at too high a rate is likely to have a catastrophic effect on throughput. Dropping at a low rate ( $< 1$  packet per round-trip-time) then increasing the drop rate slowly until the persistent queue goes below target is unlikely to overdrop yet is guaranteed to eventually dissipate the persistent queue. This stochastic gradient learning procedure is the core of CoDel's control loop (the gradient exists because a drop always reduces the (instantaneous) queue so an increasing drop rate always moves the system "down" toward no persistent queue, regardless of traffic mix).

The next drop time is decreased in inverse proportion to the square root of the number of drops since the dropping state was entered, using the well-known nonlinear relationship of drop rate to throughput to get a linear change in throughput. [REDL1998, MACTCP1997]

Since the best rate to start dropping is at slightly more than one packet per RTT, the controller's initial drop rate can be directly derived from the Estimator's interval, defined in section 3.1. Where the interval is likely to be very close to the usual round trip time, the initial drop spacing SHOULD be set to the Estimator's interval plus twice the target (i.e., initial drop spacing =  $1.1 \times \text{interval}$ ) to ensure that acceptable congestion delays are covered.

Use of the minimum statistic lets the Controller be placed in the dequeue routine with the Estimator. This means that the control signal (the drop) can be sent at the first sign of bad queue (as indicated by the sojourn time) and that the Controller can stop acting as soon as the sojourn time falls below the Setpoint. Dropping at dequeue has both implementation and control advantages.

#### 4. Putting it together: queue management for the network edge

The CoDel building blocks are able to adapt to different or time-varying link rates, to be easily used with multiple queues, to have excellent utilization with low delay and to have a simple and efficient implementation. The only setting CoDel requires is its interval value, and as 100ms satisfies that definition for normal internet usage, CoDel can be parameter-free for consumer use. CoDel was released to the open source community where it has been widely promulgated and adapted to many problems. We can see how well these building blocks work in a simple CoDel queue management implementation. This AQM was designed as a bufferbloat solution and is focused on the consumer network edge.

##### 4.1. Overview of CoDel AQM

To ensure that link utilization is not adversely affected, CoDel's Estimator sets its target to the Setpoint that optimizes power and CoDel's Controller does not drop packets when the drop would leave the queue empty or with fewer than a maximum transmission unit (MTU) worth of bytes in the buffer. Section 3.2 showed that the ideal Setpoint is 5-10% of the connection RTT. In the open Internet, in particular the consumer edge, we can use the "usual maximum" terrestrial RTT of 100 ms to calculate a minimum target of 5ms. Under the same assumptions, we compute the interval for tracking the minimum to be the nominal RTT of 100ms. In practice, uncongested links will see sojourn times under the target more often than once per RTT, so the Estimator is not overly sensitive to the value of the interval.

When the Estimator finds a persistent delay above target, the Controller enters the drop state where a packet is dropped and the next drop time is set. As discussed in section 3.3, the initial next drop spacing is intended to be long enough to give the endpoints time to react to the single drop so SHOULD be set to a value of 1.0 to 1.1 times the interval. If the Estimator's output falls below the target, the Controller cancels the next drop and exits the drop state. (The Controller is more sensitive than the Estimator to an overly short interval, since an unnecessary drop could occur and lower utilization.) If next drop time is reached while the Controller is still in drop state, the packet being dequeued is

dropped and the next drop time is recalculated. Additional logic prevents re-entering the dropping state too soon after exiting it and resumes the dropping state at a recent control level, if one exists.

Note that CoDel AQM only enters its dropping state when the local minimum sojourn delay has exceeded an acceptable standing queue target for a time interval long enough for normal bursts to dissipate ensuring that a burst of packets that fits in the pipe will not be dropped.

CoDel's efficient implementation and lack of configuration are unique features and make it suitable to manage modern packet buffers. For more background and results on CoDel, see [CODEL2012] and <http://pollere.net/CoDel.html>.

#### 4.2. Non-starvation

CoDel's goals are to control delay with little or no impact on link utilization and to be deployed on a wide range of link bandwidth, including varying rate links, without reconfiguration. To keep from making drops when it would starve the output link, CoDel makes another check before dropping to see if at least an MTU worth of bytes remains in the buffer. If not, the packet SHOULD NOT be dropped and, currently, CoDel exits the drop state. The MTU size can be set adaptively to the largest packet seen so far or can be read from the driver.

#### 4.3. Using the interval

The interval is chosen to give endpoints time to react to a drop without being so long that response times suffer. CoDel's Estimator, Setpoint, and Control Loop all use the interval. Understanding their derivation shows that CoDel is the most sensitive to the value of interval for single long-lived TCPs with a decreased sensitivity for traffic mixes. This is fortunate as RTTs vary across connections and are not known apriori and it's difficult to obtain a definitive histogram of RTTs seen on the normal consumer edge link. The best policy is to use an interval slightly larger than the RTT seen by most of the connections using a link, a value that can be determined as the largest RTT seen if the value is not an outlier (as in section 3.1, use of a 95-99th percentile value should work). In practice, this value is not known or measured (though see Section 6.2 for an application where interval is measured. Work-in-progress at Pollere may lead to a method of doing this in an Internet buffer). A setting of 100ms works well across a range of RTTs from 10ms to 1 second (excellent performance is achieved in the range from 10 ms to 300ms). For devices intended for the normal terrestrial Internet interval SHOULD have the value of 100ms. This will only cause overdropping

where a long-lived TCP has an RTT longer than 100ms and there is little or no mixing with other connections through the link.

Some confusion concerns the roles of the target Setpoint and the minimum-tracking interval. In particular, some experimenters believe the value of target needs to be increased when the lower layers have a bursty nature where packets are transmitted for short periods interspersed with idle periods where the link is waiting for permission to send. CoDel's Estimator will "see" the effective transmission rate over an interval and increasing target will just lead to longer queue delays. On the other hand, where a significant additional delay is added to the intrinsic round trip time of most or all packets due to the waiting time for a transmission, it is necessary to increase interval by that extra delay. That is, target SHOULD NOT be adjusted but interval MAY need to be adjusted. For more on this (and pictures) see <http://pollere.net/Pdfdocs/noteburstymacs.pdf>

#### 4.4. The target Setpoint

The target is the maximum acceptable standing queue delay above which CoDel is dropping or preparing to drop and below which CoDel will not drop. The calculations of section 3.2 showed that the best setpoint is 5-10% of the RTT, with the low end of 5% preferred. We used simulation to explore the impact when TCPs are mixed with other traffic and with connections of different RTTs. Accordingly, we experimented extensively with values in the 5-10% of RTT range and, overall, used target values between 1 and 20 milliseconds for RTTs from 30 to 500ms and link bandwidths of 64Kbps to 100Mbps to experimentally explore the Setpoint that gives consistently high utilization while controlling delay across a range of bandwidths, RTTs, and traffic loads. Our results were notably consistent with the mathematics of section 3.2. Below a target of 5ms, utilization suffers for some conditions and traffic loads, above 5ms we saw very little or no improvement in utilization. Thus target SHOULD be set to 5ms for normal Internet traffic.

If a CoDel link has only or primarily long-lived TCP flows sharing a link to congestion but not overload, the median delay through the link will tend to the target. For bursty traffic loads and for overloaded conditions (where it is difficult or impossible for all the arriving flows to be accommodated) the median queues will be longer than target.

The non-starvation drop inhibit feature dominates where the link rate becomes very small. By inhibiting drops when there is less than an (outbound link) MTU worth of bytes in the buffer, CoDel adapts to very low bandwidth links. This is shown in [CODEL2012] and

interested parties should see the discussion of results there. Unpublished studies were carried out down to 64Kbps. The drop inhibit condition can be expanded to include a test to retain sufficient bytes or packets to fill an allocation in a request-and-grant MAC.

Sojourn times must remain above target for an entire interval in order to enter the drop state. Any packet with a sojourn time less than target will reset the time that the queue was last below the target. Since Internet traffic has very dynamic characteristics, the actual sojourn delays experienced by packets varies greatly and is often less than the target unless the overload is excessive. When a link is not overloaded, it is not a bottleneck and packet sojourn times will be small or nonexistent. In the usual case, there are only one or two places along a path where packets will encounter a bottleneck (usually at the edge), so the amount of queuing delay experienced by a packet should be less than 10 ms even under extremely congested conditions. Contrast this to the queuing delays that grow to orders of seconds that have led to the "bufferbloat" term [NETAL2010, CHARRB2007].

#### 4.5. Use with multiple queues

Unlike other AQMs, CoDel is easily adapted to multiple queue systems. With other approaches there is always a question of how to account for the fact that each queue receives less than the full link rate over time and usually sees a varying rate over time. This is exactly what CoDel excels at: using a packet's sojourn time in the buffer completely bypasses this problem. A separate CoDel algorithm runs on each queue, but each CoDel uses the packet sojourn time the same way a single queue CoDel does. Just as a single queue CoDel adapts to changing link bandwidths[CODEL2012], so do the multiple queue CoDels. When testing for queue occupancy before dropping, the total occupancy of all bins should be used. This property of CoDel has been exploited in fq\_codel, briefly discussed in the next section and the subject of another Internet Draft.

#### 4.6. Use of stochastic bins or sub-queues to improve performance

Shortly after the release of the CoDel pseudocode, Eric Dumazet created fq\_codel, applying CoDel to each bin, or queue, used with stochastic fair queuing. (To understand further, see [SFQ1990] or the linux sfq at <http://linux.die.net/man/8/tc-sfq>.) Fq\_codel hashes on the packet header fields to determine a specific bin, or sub-queue, for each five-tuple flow, and runs CoDel on each bin or sub-queue thus creating a well-mixed output flow and obviating issues of reverse path flows (including "ack compression"). Dumazet's code is part of the CeroWrt project code at the [bufferbloat.net](http://bufferbloat.net)'s web site

and an Internet Draft has been submitted describing `fq_codel`, `draft-hoeiland-joergensen-mckenney-taht-gettys-dumazet-fq-codel`.

We've experimented with a similar approach by creating an ns-2 simulator code module, `sfq_codel`. This has provided excellent results thus far: median queues remain small across a range of traffic patterns that includes bidirectional file transfers (that is, the same traffic sent in both directions on a link), constant bit-rate VoIP-like flows, and emulated web traffic and utilizations are consistently better than single queue CoDel, generally very close to 100%. Our version differs from Dumazet's by preferring a packet-based round robin of the bins rather than byte-based DRR and there may be other minor differences in implementation. Our code, intended for simulation experiments, is available at <http://pollere.net/CoDel.html> and being integrated into the ns-2 distribution. Andrew McGregor has an ns-3 version of `fq_codel`.

Stochastic flow queuing provides better traffic mixing on the link and tends to isolate a larger flow or flows. For real priority treatment, use of DiffServ isolation is encouraged. We've experimented in simulation with creating a queue to isolate all the UDP traffic (which is all simulated VoIP thus low bandwidth) but this approach has to be applied with caution in the real world. Some experimenters are trying rounding with a small quantum (on the order of a voice packet size) but this also needs thorough study.

A number of open issues should be studied. In particular, if the number of different queues or bins is too large, the scheduling will be the dominant factor, not the AQM; it is NOT the case that more bins are always better. In our simulations, we have found good behavior across mixed traffic types with smaller numbers of queues, 8-16 for a 5Mbps link. This configuration appears to give the best behavior for voice, web browsing and file transfers where increased numbers of bins seems to favor file transfers at the expense of the other traffic. Our work has been very preliminary and we encourage others to take this up and to explore analytic modeling. It would be instructive to see the effects of different numbers of bins on a range of traffic models, something like an updated version of [BMPFQ].

Implementers SHOULD use the `fq_codel` multiple queue approach if possible as it deals with many problems beyond the reach of an AQM on a single queue.

#### 4.7. Setting up CoDel AQM

CoDel's is set for use in devices in the open Internet. An interval of 100ms is used, target is set to 5% of interval, and the initial drop spacing is also set to interval. These settings have been chosen so that a device, such as a small WiFi router, can be sold without the need for any values to be made adjustable, yielding a parameterless implementation. In addition, CoDel is useful in environments with significantly different characteristics from the normal Internet, for example, in switches used as a cluster interconnect within a data center. Since cluster traffic is entirely internal to the data center, round trip latencies are low (typically <100us) but bandwidths are high (1-40Gbps) so it's relatively easy for the aggregation phase of a distributed computation (e.g., the Reduce part of a Map/Reduce) to persistently fill then overflow the modest per-port buffering available in most high speed switches. A CoDel configured for this environment (target and interval in the microsecond rather than millisecond range) can minimize drops (or ECN marks) while keeping throughput high and latency low.

Devices destined for these environments MAY use a different interval, where suitable. If appropriate analysis indicates, the target MAY be set to some other value in the 5-10% of interval and the initial drop spacing MAY be set to a value of 1.0 to 1.2 times the interval. But these settings will cause problems such as over dropping and low throughput if used on the open Internet so devices that allow CoDel to be configured MUST default to Internet appropriate values given in this document.

#### 5. Annotated Pseudo-code for CoDel AQM

What follows is the CoDel algorithm in C++-like pseudo-code. Since CoDel adds relatively little new code to a basic tail-drop fifo-queue, we've tried to highlight just these additions by presenting CoDel as a sub-class of a basic fifo-queue base class. There have been a number of minor variants in the code and our reference pseudo-code has not yet been completely updated. The reference code is included to aid implementers who wish to apply CoDel to queue management as described here or to adapt its principles to other applications.

Implementors are strongly encouraged to also look at Eric Dumazet's Linux kernel version of CoDel - a well-written, well tested, real-world, C-based implementation. As of this writing, it is at:

```
http://git.kernel.org/?p=linux/kernel/git/torvalds/
linux.git;a=blob_plain;f=net/sched/sch_codel.c;hb=HEAD
```



This code is open-source with a dual BSD/GPL license:

Codel - The Controlled-Delay Active Queue Management algorithm

Copyright (C) 2011-2014 Kathleen Nichols <nichols@pollere.com>

Redistribution and use in source and binary forms, with or without modification, are

permitted provided that the following conditions are met:

- o Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer, without modification.
- o Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- o The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

Alternatively, provided that this notice is retained in full, this software may be distributed under the terms of the GNU General Public License ("GPL") version 2, in which case the provisions of the GPL apply INSTEAD OF those given above.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 5.1. Data Types

"time\_t" is an integer time value in units convenient for the system. Resolution to at least a millisecond is required and better resolution is useful up to the minimum possible packet time on the output link; 64- or 32-bit widths are acceptable but with 32 bits the resolution should be no finer than  $2^{-16}$  to leave enough dynamic

range to represent a wide range of queue waiting times. Narrower widths also have implementation issues due to overflow (wrapping) and underflow (limit cycles because of truncation to zero) that are not addressed in this pseudocode. The code presented here uses 0 as a flag value to indicate "no time set."

"packet\_t\*" is a pointer to a packet descriptor. We assume it has a `tstamp` field capable of holding a `time_t` and that field is available for use by CoDel (it will be set by the `enqueue` routine and used by the `dequeue` routine).

"queue\_t" is a base class for queue objects (the parent class for `codel_queue_t` objects). We assume it has `enqueue()` and `dequeue()` methods that can be implemented in child classes. We assume it has a `bytes()` method that returns the current queue size in bytes. This can be an approximate value. The method is invoked in the `dequeue()` method but shouldn't require a lock with the `enqueue()` method.

"flag\_t" is a Boolean.

#### 5.2. Per-queue state (`codel_queue_t` instance variables)

```
time_t first_above_time; // Time to declare sojourn time above target
time_t drop_next;       // Time to drop next packet
uint32_t count;         // Packets dropped since entering drop state
flag_t dropping;        // Equal to 1 if in drop state
```

#### 5.3. Constants

```
time_t target = MS2TIME(5); // 5ms target queue delay
time_t interval = MS2TIME(100); // 100ms sliding-minimum window
u_int maxpacket = 512; // Maximum packet size in bytes
// (should use interface MTU)
```

#### 5.4. Enqueue routine

All the work of CoDel is done in the `dequeue` routine. The only CoDel addition to `enqueue` is putting the current time in the packet's `tstamp` field so that the `dequeue` routine can compute the packet's sojourn time.

```
void codel_queue_t::enqueue(packet_t* pkt)
{
    pkt->tstamp() = clock();
    queue_t::enqueue(pkt);
}
```

### 5.5. Deque routine

This is the heart of CoDel. There are two branches: In packet-dropping state (meaning that the queue-sojourn time has gone above target and hasn't come down yet), then we need to check if it's time to leave or if it's time for the next drop(s); if we're not in dropping state, then we need to decide if it's time to enter and do the initial drop.

```
Packet* CoDelQueue::deque()
{
    double now = clock();
    dodequeResult r = dodeque(now);

    if (dropping_) {
        if (! r.ok_to_drop) {
            // sojourn time below target - leave dropping state
            dropping_ = 0;
        }
        // Time for the next drop. Drop current packet and dequeue
        // next. If the dequeue doesn't take us out of dropping
        // state, schedule the next drop. A large backlog might
        // result in drop rates so high that the next drop should
        // happen now, hence the 'while' loop.
        while (now >= drop_next_ && dropping_) {
            drop(r.p);
            r = dodeque(now);
            if (! r.ok_to_drop) {
                // leave dropping state
                dropping_ = 0;
            } else {
                ++count_;
                // schedule the next drop.
                drop_next_ = control_law(drop_next_);
            }
        }
        // If we get here we're not in dropping state. The 'ok_to_drop'
        // return from dodeque means that the sojourn time has been
        // above 'target' for 'interval' so enter dropping state.
    } else if (r.ok_to_drop) {
        drop(r.p);
        r = dodeque(now);
        dropping_ = 1;

        // If min went above target close to when it last went
        // below, assume that the drop rate that controlled the
        // queue on the last cycle is a good starting point to
        // control it now. ('drop_next' will be at most 'interval'
        // later than the time of the last drop so 'now - drop_next'
        // is a good approximation of the time from the last drop
        // until now.)
        count_ = (count_ > 2 && now - drop_next_ < 8*interval_) ?
            count_ - 2 : 1;
        drop_next_ = control_law(now);
    }
    return (r.p);
}
```

## 5.6. Helper routines

Since the degree of multiplexing and nature of the traffic sources is unknown, CoDel acts as a closed-loop servo system that gradually increases the frequency of dropping until the queue is controlled (sojourn time goes below target). This is the control law that governs the servo. It has this form because of the  $\sqrt{p}$  dependence of TCP throughput on drop probability. Note that for embedded systems or kernel implementation, the inverse sqrt can be computed efficiently using only integer multiplication. See Eric Dumazet's excellent Linux CoDel implementation for example code (in file `net/sched/sch_codel.c` of the kernel source for 3.5 or newer kernels).

```
time_t codel_queue_t::control_law(time_t t)
{
    return t + interval / sqrt(count);
}
```

Next is a helper routine that does the actual packet dequeue and tracks whether the sojourn time is above or below target and, if above, if it has remained above continuously for at least interval. It returns two values, a Boolean indicating if it is OK to drop (sojourn time above target for at least interval) and the packet dequeued.

```

typedef struct {
    packet_t* p;
    flag_t ok_to_drop;
} dodequeue_result;

dodequeue_result codel_queue_t::dodequeue(time_t now)
{
    dodequeueResult r = { NULL, queue_t::deque() };
    if (r.p == NULL) {
        // queue is empty - we can't be above target
        first_above_time_ = 0;
        return r;
    }

    // To span a large range of bandwidths, CoDel runs two
    // different AQMs in parallel. One is sojourn-time-based
    // and takes effect when the time to send an MTU-sized
    // packet is less than target. The 1st term of the "if"
    // below does this. The other is backlog-based and takes
    // effect when the time to send an MTU-sized packet is >=
    // target. The goal here is to keep the output link
    // utilization high by never allowing the queue to get
    // smaller than the amount that arrives in a typical
    // interarrival time (MTU-sized packets arriving spaced
    // by the amount of time it takes to send such a packet on
    // the bottleneck). The 2nd term of the "if" does this.
    time_t sojourn_time = now - r.p->tstamp;
    if (sojourn_time_ < target_ || bytes() <= maxpacket_) {
        // went below - stay below for at least interval
        first_above_time_ = 0;
    } else {
        if (first_above_time_ == 0) {
            // just went above from below. if still above at
            // first_above_time, will say it's ok to drop.
            first_above_time_ = now + interval_;
        } else if (now >= first_above_time_) {
            r.ok_to_drop = 1;
        }
    }
    return r;
}

```

## 5.7. Implementation considerations

Since CoDel requires relatively little per-queue state and no direct communication or state sharing between the enqueue and dequeue routines, it's relatively simple to add it to almost any packet processing pipeline, including ASIC- or NPU-based forwarding engines.

One issue to think about is dedequeue's use of a 'bytes()' function to find out about how many bytes are currently in the queue. This value does not need to be exact. If the enqueue part of the pipeline keeps a running count of the total number of bytes it has put into the queue and the dequeue routine keeps a running count of the total bytes it has removed from the queue, 'bytes()' is just the difference between these two counters. 32 bit counters are more than adequate. Enqueue has to update its counter once per packet queued but it doesn't matter when (before, during or after the packet has been added to the queue). The worst that can happen is a slight, transient, underestimate of the queue size which might cause a drop to be briefly deferred.

## 6. Adapting and applying CoDel's building blocks

CoDel is being implemented and tested in a range of environments. Dave Taht has been instrumental in the integration and distribution of bufferbloat solutions, including CoDel, and has set up a website and a mailing list for CeroWRT implementers. ([www.bufferbloat.net/projects/codel](http://www.bufferbloat.net/projects/codel)) This is an active area of work and an excellent place to track developments.

### 6.1. Validations and available code

An experiment by Stanford graduate students successfully used the linux CoDel to duplicate our published simulation work on CoDel's ability to following drastic link rate changes which can be found at: <http://reproducingnetworkresearch.wordpress.com/2012/06/06/solving-bufferbloat-the-codel-way/> .

Our ns-2 simulations are available at <http://pollere.net/CoDel.html> . Cable Labs has funded some additions to the simulator sfq\_codel code which have been made public. The basic algorithm of CoDel remains unchanged, but we continue to experiment with drop interval setting when resuming the drop state, inhibiting or canceling drop state when bytes in the queue small, and other minor details. Our approach to changes is to only make them if we are convinced they do more good than harm, both operationally and in the implementation. With this in mind, some of these issues may continue to evolve as we get more deployment and as the building blocks are applied to a wider range of problems.

CoDel is being made available with the ns-2 distribution.

Andrew McGregor has an ns-3 implementation of both CoDel and FQ\_CoDel (<https://github.com/dtaht/ns-3-dev>) .

CoDel is available in Linux. Eric Dumazet has put CoDel into the Linux distribution.

## 6.2. CoDel in the datacenter

Nandita Dukkkipati's team at Google was quick to realize that the CoDel building blocks could be applied to bufferbloat problems in datacenter servers, not just to Internet routers. The Linux CoDel queueing discipline (Qdisc) was adapted in three ways to tackle this bufferbloat problem.

Non-data packets were not dropped as these are typically small and sometimes critical control packets. Being located on the server, there is no concern with misbehaving users scamming such a policy as there would be in an Internet router.

In several data center workload benchmarks, which are typically bursty, CoDel reduced the queueing latencies at the Qdisc, and thereby improved the mean and 99 percentile latencies from several tens of milliseconds to less than one millisecond. The minimum tracking part of the CoDel framework proved useful in disambiguating "good" queue versus "bad" queue, particularly helpful in controlling Qdisc buffers that are inherently bursty because of TCP Segmentation Offload.

## 7. Security Considerations

This document describes an active queue management algorithm for implementation in networked devices. There are no specific security exposures associated with CoDel.

## 8. IANA Considerations

This document does not require actions by IANA.

## 9. Conclusions

CoDel provides very general, efficient, parameterless building blocks for queue management that can be applied to single or multiple queues in a variety of data networking scenarios. It is a critical tool in solving bufferbloat. CoDel's settings MAY be modified for other special-purpose networking applications.

On-going projects are creating a deployable CoDel in Linux routers and experimenting with applying CoDel to stochastic queuing with very promising results.



## 10. Acknowledgments

The authors wish to thank Jim Gettys for the constructive nagging that made us get the work "out there" before we thought it was ready. We also want to thank Dave Taht, Eric Dumazet, and the open source community for showing the value of getting it "out there" and for making it real. We also wish to thank Nandita Dukkhipati for contribution to section 6 and for comments which helped to substantially improve this draft.

## 11. References

### 11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 11.2. Informative References

- [TSV2011] Gettys, J., "Bufferbloat: Dark Buffers in the Internet", IETF 80 presentation to Transport Area Open Meeting, March, 2011, <<http://www.ietf.org/proceedings/80/tsvarea.html>>.
- [BB2011] Gettys, J. and K. Nichols, "Bufferbloat: Dark Buffers in the Internet", Communications of the ACM 9(11) pp. 57-65, .
- [BMPFQ] Suter, B., "Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-flow Queueing", IEEE Journal on Selected Areas in Communications Vol. 17 Issue 6, June, 1999, pp. 1159-1169, .
- [CMNTS] Allman, M., "Comments on Bufferbloat", Computer Communication Review Vol. 43 No. 1, January, 2013, pp. 31-37, .
- [CODEL2012] Nichols, K. and V. Jacobson, "Controlling Queue Delay", Communications of the ACM Vol. 55 No. 11, July, 2012, pp. 42-50, .
- [VANQ2006] Jacobson, V., "A Rant on Queues", talk at MIT Lincoln Labs, Lexington, MA July, 2006, <<http://www.pollere.net/Pdfdocs/QrantJul06.pdf>>.

- [REDL1998] Nichols, K., Jacobson, V., and K. Poduri, "RED in a Different Light", Tech report, September, 1999, <[http://www.cnaf.infn.it/~ferrari/papers/ispn/red\\_light\\_9\\_30.pdf](http://www.cnaf.infn.it/~ferrari/papers/ispn/red_light_9_30.pdf)>.
- [NETAL2010] Kreibich, C., et. al., "Netalyzr: Illuminating the Edge Network", Proceedings of the Internet Measurement Conference Melbourne, Australia, 2010, .
- [TSV84] Jacobson, V., "CoDel talk at TSV meeting IETF 84", <<http://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf>>.
- [CHARB2007] Dischinger, M., et. al, "Characterizing Residential Broadband Networks", Proceedings of the Internet Measurement Conference San Diego, CA, 2007, .
- [MACTCP1997] Mathis, M., Semke, J., and J. Mahdavi, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM SIGCOMM Computer Communications Review Vol. 27 no. 1, Jan. 2007, .
- [SFQ1990] McKenney, P., "Stochastic Fairness Queuing", Proceedings of IEEE INFOCOMM 90 San Francisco, 1990, .
- [KLEIN81] Kleinrock, L. and R. Gail, "An Invariant Property of Computer Network Power", International Conference on Communications June, 1981, <<http://www.lk.cs.ucla.edu/data/files/Gail/power.pdf>>.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.
- [RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

Authors' Addresses

Kathleen Nichols  
Pollere, Inc.  
PO Box 370201  
Montara, CA 94037  
USA

Email: [nichols@pollere.com](mailto:nichols@pollere.com)

Van Jacobson  
Google

Email: [vanj@google.com](mailto:vanj@google.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: April 25, 2013

J. Pouwelse, Ed.  
Delft University of Technology  
October 22, 2012

Media without censorship (CensorFree) scenarios  
draft-pouwelse-censorfree-scenarios-02

## Abstract

This document describes some scenarios in which one can imagine that the ability of authoritarian regime to censor news dissemination is reduced. It tries to draw some conclusions about what's desirable and what's not acceptable for users in those scenarios.

The CensorFree objective is to standardize the protocols for microblogging on smartphones with a focus on security and censorship resistance. Microblog entries are short text messages, possibly enriched with pictures or streaming video. The goal is to devise protocols which guard against all known forms of censorship such as: cyberspace sabotage, digital eavesdropping, infiltration, fraud, Internet kill switches and lawyer-based attacks with the best known protective methods.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents  
(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Requirements Language . . . . .	3
2. Introduction . . . . .	3
3. Goal: microblogging . . . . .	4
4. Driving scenarios . . . . .	5
4.1. 20sec scenario . . . . .	6
4.1.1. Adversary model: A simplistic attacker . . . . .	6
4.1.2. Scenario details and architectural requirements . . . . .	6
4.2. Kill-switch scenario . . . . .	8
4.2.1. Adversary model: An advanced attacker . . . . .	8
4.2.2. Scenario details and architectural requirements . . . . .	8
4.3. friend-to-friend scenario . . . . .	9
4.3.1. Adversary model: A powerful attacker . . . . .	9
4.3.2. Scenario details and architectural requirements . . . . .	10
4.4. Transmorph ability . . . . .	10
4.5. A single global conversation . . . . .	11
4.6. Spammers and hoaxes . . . . .	11
5. Design principles: simplicity and prior success . . . . .	12
6. Background rant: lack of coordination and fragmentation . . . . .	12
7. Current running code and related work . . . . .	13
8. Open issues . . . . .	14
8.1. Use cases and threat model . . . . .	15
8.2. System components, definitions and system architecture . . . . .	15
8.3. Current technology and gap . . . . .	15
8.4. Detailed system design and protocol specification . . . . .	15
9. Security Considerations . . . . .	15
10. IANA Considerations . . . . .	15
11. References . . . . .	15
11.1. Normative References . . . . .	15
11.2. Informative References . . . . .	15
11.3. URL References . . . . .	15

## 1. Requirements Language

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].

## 2. Introduction

Bits moving across the Internet are vulnerable to surveillance and censorship on an unprecedented scale. Today, both Internet providers and governments possess the ability to monitor the moves of their digital citizens from central infrastructure points. This monitoring ability creates significant potential for abuse, and a threat that goes beyond the scope of mere monitoring or filtering.

Governments have demonstrated their ability to disable communications networks in times of crisis. During the 2011 Arab Spring, Egyptian authorities demanded that telecommunication companies sever their broadband connections and mobile networks--both local and European operators were forced to comply, and, as a result, digital Egypt vanished. Despite the country's decentralized infrastructure, an Internet blackout was relatively easy to carry out. The roles and consequences of social media (e.g., Facebook and Twitter) during that same period further illustrate the capacity governments have for Internet censorship and the challenges activists face in combating it. The April 6 Youth Movement from Egypt committed digital dissent in full public view. According to The New York Times[YOUTH], the movement "provided a structure for a new generation of Egyptians, who aren't part of the nation's small coterie of activists and opinion makers, to assemble virtually and communicate freely about their grievances." But moving protest organizations to social media accessible to the public-at-large can hold surprising risks. On the ground, the movement's organization of labor strikes and protests in Facebook groups, many with thousands of followers, triggered arrest and imprisonment. Protesters in other countries quickly took note of Egypt's lesson and disabled their public Facebook profiles. In response, one government initiated social media searches on incoming, young, plane travelers by forcing them to login to Facebook upon arrival, thereby revealing online activities and any anti-government sympathies.[FORCEDLOGIN]

A glimmer of hope exists. The Arab Spring shows that a new generation is claiming their right to express themselves. Microblogging, social media in general and traditional satellite news broadcast networks are perceived as critical catalysts for political change. Generic computational fabric is soon getting in the hands of two billion people with the growth of smartphones and increasingly affordable communication. These smartphones are increasingly used to

record and spread disruptive audiovisual material, even in regions without media freedom.

Democratic countries also face a dilemma. Restrictions on the free information flow is the topics of several proposed laws by elected representatives. The strength of copyright law impacts digital information flow. Politicians must decide between weak copyright law, as championed by civil rights activists versus strong copyright enforcement, as promoted by numerous players in the creative industries. Recent furor around SOPA, PIPA, etc. in the US plus the European Parliament vote on ACTA is highly relevant in this context.

The uniqueness of The Internet lies in the IETF standards. Moving certain bits to certain locations or offering a service requires no prior official approval. However, Internet-deployed mechanisms now exist which filter news and media in general for both surveillance and censorship. The Internet has ceased to provide reliable transport service for all users. The IETF can repeat it's historical inter-networking role again by setting the standard for reliable flow of packets of news.

### 3. Goal: microblogging

The goal is creating a microblogging standard and facilitating a reference implementation for portable devices which is capable of operating in a hostile environment. This standard should be resilient against a government Internet kill switch. Microblogging is an increasingly popular technology for lightweight interaction over the Internet. It differs from traditional blogging in that [OPENMICRO]:

- o Posts are short (typically less than 140 characters, which is the limit in SMS).
- o Posts are in plain text.
- o People can reply to your posts, but not directly comment on them.
- o People learn about your posts only if they have permission to view them.
- o Your microblogging feed is discovered based on your identity at a domain or with a service.

This proposed draft standard SHALL provide: "information dissemination from a single smartphone to an audience of millions in the form of microblogging, enriched with pictures or streaming video which is guarded against all known forms of censorship such as:

cyberspace sabotage, digital eavesdropping, infiltration, fraud, Internet kill switches and lawyer-based attacks with the best known protective methods".

The focus on microblogging is driven by feasibility. Creating a standard for overcoming censorship for social networks, search engines or web browsing in general is extremely challenging. Mitigating the threats posed by Internet kill switches requires focus on the most feasible viable standard. The related work listed in this document shows existing operational systems. Existing systems cover all functionality we desire, however none of them cover all aspects and little interoperability exists.

As early as 2006, long before Arab spring events, it was reported that individuals in wide swathes of the Arab world were using Bluetooth technology to bypass police restrictions. According to news reports[DATING], communication between men and women in this region had been made possible by cellphone technology. When Bluetooth-capable phones are in close proximity, they can engage directly in digital and social chatter--no other infrastructure is needed. Moreover, when sharing photos or bandwidth-hungry videos with friends it also pays to be close. Government provided cellphone networks might not be filtering you, but can still be dreadfully slow. It therefore pays to use cell phones' Bluetooth-based, direct file-transfer features--and it comes as no surprise that wireless-transfer apps have seen millions of installs. A query of Google Trends for the phrase "Bluetooth transfer" reveals a geographical spread of this interesting social phenomenon[TREND]. It seems millions of mobile phone owners are already employing the social practice of wireless data exchange. Viability is increased by building upon this practice.

#### 4. Driving scenarios

Recent Arab spring events have shown the power of ubiquitous camera-phones, new media and microblogging. This document proposes to use smartphones, wifi and USB sticks for multimedia transport and playback. The architecture, features and driving scenarios are specifically crafted to enable compliant implementations as a single smartphone app without any additional server infrastructure.

Each scenario is focused on certain threats in a hostile environment. The adversary becomes stronger in several of the following scenarios and we also focus on the social media context.



#### 4.1. 20sec scenario

First scenario, called "20sec", defines an open microblogging standard. This first scenario duplicates existing microblogging practices with an open standard in a fully decentralized setting. The scenario requirements are performance equal to central-server based approach (e.g. the ability to reach 20 million people in 20 seconds).

##### 4.1.1. Adversary model: A simplistic attacker

Eavesdropping is a common and easy passive attack in a hostile environment. In this scenario we assume the attacker has full access to the network between the user and any Internet server. Specifically, the adversary can observe, block, delay, replay and modify all traffic coming from any server. Furthermore, all servers such as DNS servers, web servers, swarm trackers, CDN cloud servers and access portals are assumed to be under direct or indirect control of the adversary.

The adversary cannot compromise traffic between smartphones or other participating devices. The adversary cannot compromise smartphones or other participating devices. The adversary cannot break standard cryptographic primitives, such as block ciphers and message-authentication codes.

##### 4.1.2. Scenario details and architectural requirements

Smartphone owner Alice with wifi-based Internet access records an eye witness video. She attaches this video to a microblog entry and shares this story plus video automatically with friends Bob and Charlie which are subscribed to her news feed. Alice does not need to trust any central server with her credentials or has to prove her identity to a central (web) server. Bob and Charlie are both behind a NAT middlebox compliant to the BEHAVE recommendations [RFC4787]. No assistance of a coordinating server (e.g. STUN or TURN) is required to traverse this NAT box using UDP messages. This scenario assumes direct or NAT-based Internet access (the next scenario deals with packet forwarding).

Performance should be equal to central-server based approach, providing the ability to reach 20 million people in 20 seconds. This first scenario duplicates existing microblogging practices with an open standard in a fully decentralized setting. The 20sec scenario requires that solutions provide seamless backwards compatibility with existing leading solutions (e.g. Twitter, Sina Weibo, chyrp, heello) by using content import tools. Proposed open solutions MUST permit easy bulk trans-coding and ingest of existing news feeds into this

open standard.

An essential feature of the 20sec scenario is all central gatekeepers or communication to them is possibly compromised. Ownership of data is fundamental to autonomy. To meet the anti-censorship goal, 20sec assumes an infrastructure which is not dependent and completely decoupled from potentially hostile servers such as DNS servers and web servers. 20sec MUST be based on full self-organization. The infrastructure consists purely of devices running compliant implementations. No central server requires installation or maintenance, making this infrastructure independent on any type of funding or business model. 20sec requires an overlay which is highly resilient. Smartphones, tablets and PCs are able to utilize this P2P overlay for microblogging. Existing solutions such as [OPENMICRO] require a central webserver and OAuth-like authentication primitives. This prior work is not suitable for our 20sec scenario, as we aim to remove all server, ultrapeer or superpeer reliance and equality of all participants in the overlay.

When Alice downloads her smartphone app and starts it for the first time it needs to bootstrap. On this initial startup, the microblogging software must bootstrap and find at least one other peer in the overlay. The most simple method of bootstrapping is using a list of currently online peers plus their port number. See the example below.

```
# file: Central-Bootstrap-Servers.txt
# default bootstrap peers
server1.always-online.org 6420
host1.never-offline.ro 6420
sealand.routed.org 6420
168.0.0.13 6420
```

A file sharing program needs a fresh list of peers to bootstrap. Thus a pre-defined list of peers is included in the software installer. As peers can go offline it is important that at least one peer out of possibly thousands on the list is still online. This pre-existing address list of possibly working peers must therefore remain valid for as long as possible. Bootstrapping is done by contacting peers in the list, possibly in parallel. If a single peers replies, the smartphone app of Alice is connected. Once connected, a fresh list of working peer Internet addresses COULD be requested. Several ideas have been proposed on bootstrapping systems without an "online bootstrap server" list. For instance, simply by smart brute force pinging, as described by the University of Denver [BOOTSTRAP].

It is RECOMMENDED compliant implementations explore and implement

efficient alternatives for decentralized initial bootstrapping.

#### 4.2. Kill-switch scenario

This scenario describes a situation without any Internet access. We assume the government has essentially "killed" the Internet, in an Arab spring like scenario. It is focused on ad-hoc packet forwarding between smartphones.

##### 4.2.1. Adversary model: An advanced attacker

The adversary has disabled all Internet-based communication.

We assume the adversary cannot eavesdrop, jam, delay, replay, modify or spoof wireless communication between smartphones. The adversary cannot compromise smartphones or other participating devices. The adversary cannot break standard cryptographic primitives, such as block ciphers and message-authentication codes.

##### 4.2.2. Scenario details and architectural requirements

Smartphone owner Alice has no Internet access. She records a video, attaches this video to a microblog entry in her phone app. Friends Bob and Charlie are subscribed to her news feed. Bob and Charlie are at some point within range of the wifi, bluetooth or other wireless capability of Alice. This fresh microblog entry plus video is shared automatically. Bob obtained the message from Alice using a smartphone app which is periodically scanning if other devices are around and if they possibly have fresh news. This periodic synchronization SHOULD be energy-efficient. Bob sees no noticeable decrease in battery lifetime after he obtained unconstrained news access. Charlie later goes to a square where numerous people have gathered, most of which are highly interested in the latest videos. The fresh messages automatically spreads in this crowd.

Note that this scenario differs from Delay-Tolerant Networking (DTN), as being investigated by a Working Group within the Internet Research Task Force [RFC4838] and scientists[BUBBLE]. The DTN focus is on finding routes to an explicitly given destination, usually by maintaining routing tables. Their system model and terminology cannot be applied in our context, for instance, "Endpoint Identifiers" which identify the original sender and final destination. In our Internet-Free scenario sender Alice does NOT explicitly send a message with destination Bob.

A wealth of related work exists in this area. General solutions are found in mobile ad hoc networks (MANET), which provide self-organized IP routing among wireless devices, and delay-tolerant networks (DTN),

which use a simple store-and-forward primitive to communicate over heterogeneous links. Mobile ad hoc networks have been studied within the Internet Research Task Force (IRTF) since 1997, leading to several standards published by the IETF's MANET Working Group, while delay-tolerant networks are currently the focus of the IRTF's DTN Research Group. We hope that much of that knowledge can be reused, despite our scenario differing slightly from DTN (as being investigated by the IRTF [RFC4838])

#### 4.3. friend-to-friend scenario

This third scenario uses friend-to-friend networking to remove the requirement for active networking and wifi sensing. The smartphones of Alice and Bob need to be synced manually. This scenario SHOULD deliver a privacy-by-design type of microblogging service.

##### 4.3.1. Adversary model: A powerful attacker

We must assume from the Arab Spring scenario the existence of a powerful adversary. For instance, the adversary has disabled all Internet-based communication. The adversary even actively monitors wireless communication. Protocol designers have identified the following threats [BRIAR] for similar circumstances:

- o The adversary can observe, block, delay, replay, and modify traffic on the underlying network. Thus, the microblogging service must ensure end-to-end security without relying on the security of the underlying network.
- o Wireless communication is regularly monitored. Responding to any wireless requests from a stranger is a direct threat to the user and extremely harmful.
- o Possession of encrypted electronic messages or encryption technology in general is extremely harmful to the smartphone owner.
- o The adversary has a limited ability to compromise smartphones or other participating devices. If a device is compromised, the adversary can access any information held in the device's volatile memory or persistent storage.
- o The adversary can choose the data written to the microblogging layer by higher protocol layers.
- o The adversary cannot break standard cryptographic primitives, such as block ciphers and message-authentication codes.

Encryption is not a sufficient requirement of the friend-to-friend scenario, everything MUST be hidden. Possession of smartphones apps with encryption is already dangerous for the owner.

#### 4.3.2. Scenario details and architectural requirements

Reports from repressive regions indicate that USB sticks are commonly used to transport sensitive information. See for instance this extensive report on North-Korea [NKOREA]. In the friend-to-friend scenario a network of friends is trusted to transport news manually, by simply carrying it around. Smartphones with NFC capability or manual USB transfer are used to duplicate and move messages. Thus Alice delivers her fresh news message to Bob, which is later given manually to Charlie.

As direct social connections are sparse and proximity of friends is not continuous, this scenario SHOULD facilitate usage of friends-of-friends or further removed social ties to relay news messages. This requires the development of a decentralized social network, for instance, with digital signatures of friendship certificates. In effect this would create a "decentralized social network", completely autonomous and owned by all participants. We assume Alice only has Bob in her friendlist and Bob only has Charlie in his friendlist. An OPTIONAL feature is that the smartphone apps running on the smartphone Alice and Charlie detect that they have friendship path through Bob. Fresh news is thus exchanged.

The interception of a single smartphone MUST NOT expose the app itself, any friend list or worst: the entire social network. We assume Alice is placing herself in danger with electronic tools for "subversive activities against the democratic republic". Information hiding techniques are essential or even life-critical. Possibly based on Zero-Knowledge Proof (ZKP) protocols [ZEROKNOW]. The smartphone app MUST pose as a harmless entertainment feature of a smartphone or use another mechanism to become a "stealth app".

This scenario requires modification and enhancement based on real-world experience from human rights activist [EGYPTSTUDY].

#### 4.4. Transmorph ability

Prior scenarios expanded the threat model. This and the following scenarios are focused on the social media context. News is created in a region without freedom and then needs to travel to the outside world. We refer to this simply as the freedom/non-freedom border. Different transport protocols, dynamics and different solutions are needed on the two sides of this border.

We now expand the friend-to-friend scenario with a transmorph ability, the ability of news to cross the freedom/non-freedom border.

Alice is a well known blogger in an region with extreme censorship. Her identity on Twitter has millions of followers. However, she has no direct ability to reach a Twitter.com server or Internet in general. We assume Alice only has Bob in her friendlist and Bob only has Charlie in his friendlist. Charlie is able to smuggle a collection of messages out of the country. The messages originating from Alice should be transmorphed into a series of Twitter post belonging to her.

The identities used in Twitter are highly identifiable labels, with a certain trust level. This hard identity with millions of followers is a stark contrasts with anonymity. Current anti-censorship technology lacks the ability to first have stealth encrypted transport of news, cross the freedom/non-freedom border and then transmorph this news into a public accessible form with a highly identifiable label.

#### 4.5. A single global conversation

Existing technologies, such as [TOR] in combination with XMPP or the Orbot smartphone app facilitate protected point-to-point communication. However, a desired scenario is to facilitate more current the Twitter-like social media practices, best typified as a "global conversation".

Furthermore, current social media revolves around video-rich, real-time interaction with groups, hashtag-based discovery and social networking. All of these aspects are not offered or are incompatible with current-generation of privacy enhancing technology. More knowledge is needed about reputation models in news reporting and information flows. In the current microblogging age, does the number of real-person followers be seen as your reputation? Do several news sources of moderate reputation which report the same news story yield together an increased reputation score?

This work should combine privacy enhancement with microblogging.

#### 4.6. Spammers and hoaxes

This final scenario is focused on spam. All technology addressing one of the above scenarios MUST also have the capability to deal with spam. Unfortunately, this ability to deal with spam is in conflict with simplicity.

Alice and Bob are exchanging the fresh messages from their social

network (similar to Internet-free or Friends-only). Eve is actively trying to disrupt the system by injecting news channels with a mix of genuine news, obviously fake messages (consuming valuable system resources and user attention) and hoaxes. These falsehoods made to masquerade as truth result in erosion of overall trust in the system.

Systems SHOULD offer capabilities to report spam, mechanisms for fact validation and reputations of (pseudo) identities.

#### 5. Design principles: simplicity and prior success

Designing and crafting software which is completely self-organizing has clear limits [CAPLIMIT] and requires a certain level of expertise [LEVELS]. In order to avoid repeating mistakes from the past, this document aims to base itA's design principles on existing new media successes. For microblogging this means following market leading solutions and enhance them with censorship resilience. We recognize the following success factors: Simplicity, Real-time responsiveness, Near-effortless news creation, News items are bundled in channels, combine public broadcasting and person-to-person private messaging, following a channel is single direction, more followers yields more visibility, keyword search with push of updates and ability to deal with spam.

#### 6. Background rant: lack of coordination and fragmentation

Computers communicating on equal footing has been part of the IETF standards for many decades. Recently several loosely connected standard initiated around explicitly driven by the P2P paradigm for applications such as Internet telephony video streaming. An essential problem in this domain is the lack of coordination and standard setting for P2P technology. A large part of the innovation around P2P seems to happen in single-person Open Source projects and small groups which lack the engineering capacity to make generic, re-usable and documented components. Given their running code-driven nature, money and time is not available for attending standards-setting meetings, writing formal specifications and defining quality control testing suites. Profit-driven organizations should have the resources to overcome these resource shortage issues. However, due to the dynamic, disruptive and litigious nature of P2P few examples exist of companies which are capable of supporting an IETF standard setting activity for several years.

As presented during IETF 81 area directorate, there is "not a clear long-term architecture yet for you to build actual classes of P2P applications using IETF technologies". Forming an overlay is hard and scalable privacy-preserving unstructured search solutions are only barely out of the scientific research community.

From the above we conclude that a key obstacle to the success of this proposal is implementation and uptake. A draft document, active community and reference implementation ideally evolve together over time. To overcome this issue a continuous incremental improvement approach is advised. The preferred way is incremental development of single a reference implementation, based on free software.

## 7. Current running code and related work

DISCLAIMER: this section needs significant expansion and listing of projects with running code and self-organization.

Several Open Source projects have running code and partially implemented the above four scenarios. We will briefly list them here.

[TOR] A free software implementation of second-generation onion routing, a system enabling its users to communicate anonymously on the Internet. This flagship project has boosted online anonymity for over a decade and the key example for the cat and mouse dynamics. The Orbot project provides an Android implementation of Tor. Due to the usage of the client server/model, exit node principle plus lack of reputations this architecture is not compatible with our scenarios.

[DIASPORA] A free personal web server that implements a distributed social networking service. This partially operational system is based on the client/server model and not compatible with our ad-hoc scenarios.

[BRIAR] Briar is a secure news and discussion system designed to be used by journalists, activists and civil society groups in authoritarian countries. Briar differs from existing circumvention tools and mesh networks in three significant ways: needs no external infrastructure, can operate over any mixture of available media and builds on social relationships. The aims of this project are similar to our scenarios, but this project lacks running code and has few active developers.

[BUBBLE] DTN researchers have simulated closely related scenarios. Dissemination in the Arab Spring scenario is likely to involve an explicit copy between people who trust each other, referred to as social-based forwarding in this study.

[TWIMIGHT] The Twimight project by ETH-Zurich university shows that decentralized microblogging already exists. Researchers developed an Android application that uses Twitter servers in normal conditions, but switches to a Bluetooth-based disaster mode when Internet



connectivity is lost.

[MUSUBI] The Musubi smartphone app represents another key, censorship-free, technology advancement. Developed by Stanford University, it offers instant messaging service and media sharing capabilities similar to WhatsApp, Ping, and Blackberry Messenger. What makes it unique is that all data and processing resides on the smartphones, not in the cloud. This decentralization removes the need for central processing and provides significant decoupling from the underlying infrastructure. Exchange of cryptographic keys is integrated in the friending process--Musubi essentially builds a decentralized social graph. Unfortunately, Musubi is also limited--all data transfers go through central servers, as it lacks NAT-traversal capability.

[TRIBLER] DISCLAIMER2: this project is coordinated by the author. This project has created Open Source firmware for a Samsung Internet-connected television which gives it the ability to find, share and stream news videos within a fully self-organizing overlay; operated only by remote control [REBELLIONTV]. It is also available as generic zero-server file sharing software for the PC which has been installed by 1.2 million users. It uses the Dispersy elastic database for providing: keyword search, content discovery, content voting and spam prevention using crowd sourcing [DISPERSY]. For swarm-based streaming and generic message transport it uses the IETF protocol developed within the PPSP working group, called Libswift [LIBSWIFT]. All this code is created by a single team and specifically designed to facilitate evolution into the prior described scenarios. An Libswift demo streaming app is available on the Android market.

## 8. Open issues

Deliverables planned and issues which need to be addressed.

TODO: ADD REF Privacy definition:

<http://tools.ietf.org/html/draft-iab-privacy-terminology-01>

TODO: REF

<http://www.ietf.org/id/draft-iab-privacy-considerations-03.txt>

TODO: <http://datatracker.ietf.org/doc/search/> P2P

TODO: <http://datatracker.ietf.org/doc/rfc4981/> SEARCH survey

TODO: <https://datatracker.ietf.org/doc/draft-ietf-p2psip-reload/>

8.1. Use cases and threat model

8.2. System components, definitions and system architecture

8.3. Current technology and gap

8.4. Detailed system design and protocol specification

9. Security Considerations

tbd.

10. IANA Considerations

tbd.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

11.2. Informative References

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

11.3. URL References

[YOUTH] <http://www.nytimes.com/2009/01/25/magazine/25bloggers-t.html>, "Revolution, Facebook-Style".

[FORCEDLOGIN] <http://online.wsj.com/article/SB125978649644673331.html>, "Iranian Crackdown Goes Global".

[NKOREA] [http://audiencescapes.org/sites/default/files/Report\\_Summary\\_Quiet\\_Opening\\_North%20Korea\\_InterMedia.pdf](http://audiencescapes.org/sites/default/files/Report_Summary_Quiet_Opening_North%20Korea_InterMedia.pdf), "A QUIET OPENING: NORTH KOREANS IN A CHANGING MEDIA ENVIRONMENT".

- [EGYPTSTUDY] <http://conferences.sigcomm.org/imc/2011/docs/pl.pdf>, "Analysis of country-wide internet outages caused by censorship".
- [OPENMICRO] <http://xmpp.org/extensions/xep-0277.html>, "XEP-0277: Microblogging over XMPP".
- [DATING] <http://www.washingtonpost.com/wp-dyn/content/article/2006/08/05/AR2006080500930.html>, "Saudi Youth Use Cellphone Savvy To Outwit the Sentries of Romance".
- [TREND] <http://www.google.com/trends/?q=bluetooth+transfer>, "Google Trends query".
- [BOOTSTRAP] <http://grothoff.org/christian/dasp2p.pdf>, "Bootstrapping Peer-to-Peer Networks".
- [ZEROKNOW] [http://www.cse.ust.hk/~liu/luli/PT\\_Trans\\_final.pdf](http://www.cse.ust.hk/~liu/luli/PT_Trans_final.pdf), "Pseudo trust: Zero-knowledge based authentication in anonymous peer-to-peer protocols".
- [CAPLIMIT] <http://doi.ieeecomputersociety.org/10.1109/MC.2012.54>, "The CAP Theorem's Growing Impact".
- [LEVELS] <http://blog.incubaid.com/2012/03/28/the-game-of-distributed-systems-programming-which-level-are-you/>, "The Game of Distributed Systems Programming. Which Level Are You?".
- [TOR] <http://www.torproject.org>, "Tor Project: Anonymity Online".
- [DIASPORA] <http://diasporaproject.org/>, "Diaspora is a fun and creative community that puts you in control.".
- [BRIAR] <https://fulpool.org/btp.pdf>, "Secure communication over diverse transports".
- [BUBBLE] <http://dx.doi.org/10.1109/TMC.2010.246>, "BUBBLE Rap: Social-Based Forwarding in Delay-Tolerant Networks".
- [TWIMIGHT] <http://dl.acm.org/citation.cfm?id=2159576.2159601>, "Twitter in disaster mode: smart probing for opportunistic peers".
- [MUSUBI] <http://dl.acm.org/citation.cfm?id=2187866>, "Musubi: disintermediated interactive social feeds for mobile devices".

- [TRIBLER] <http://dl.acm.org/citation.cfm?id=2206767>, "Tribler: P2P search, share and stream".
- [REBELLIONTV] <http://www.tribler.org/trac/wiki/SwiftTV>, "RebellionTV a.k.a. Libswift on a television project".
- [DISPERSY] [www.frayja.com/pub/dispersypaper2012.pdf](http://www.frayja.com/pub/dispersypaper2012.pdf):donotdistribute, "Dispersy elastic database".
- [LIBSWIFT] <http://www.libswift.org>, "IETF PPSP streaming protocol implementation".

## Author's Address

Johan Pouwelse (editor)  
Delft University of Technology  
Mekelweg 4  
Delft  
The Netherlands

Phone: +31 15 278 2539  
EMail: J.A.pouwelse@tudelft.nl

