Network Working Group
Internet-Draft
Expires: May 9, 2007

A. Falk
Y. Pryadkin
ISI
D. Katabi
MIT
November 5, 2006

Specification for the Explicit Control Protocol (XCP)
draft-falk-xcp-spec-02.txt

Status of this Memo

Copyright Notice

Abstract

This document contains an initial specification for the Explicit Control Protocol (XCP), an experimental congestion control protocol. XCP is designed to deliver the highest possible end-to-end throughput over a broad range of network infrastructure, including links with very large bandwidth-delay products, which are not well served by the current control algorithms. XCP is potentially applicable to any

transport protocol, although initial testing has applied it to TCP in particular. XCP routers are required to perform a small calculation on congestion state carried in each data packet. XCP routers also periodically recalculate the local parameters required to provide fairness. On the other hand, there is no per-flow congestion state in XCP routers.

This version specification (-02) includes protocol changes that move per-packet divisions from the router to the sender.

Table of Contents

1. Changes Since Last Version

Changes between version -01 and -02

- Minor edits and typo fixes.

Changes between version -00 and -01

- Updated protocol to reflect movement of the per-packet division from the router to the end-system.
- Incremented version number (to 0x02) to reflect change in packet header format.
- Reordered the Protocol, Length, Version, and Format fields in the congestion header (in anticipation of future support of IPv6 extension headers).
- Routers now MUST (from SHOULD) ignore fields other than reverse_feedback when minimal header is used.
- No longer ignore packets with RTT set to zero. Senders with coarse-grained timer may generate these if the RTT is less than the timer precision.
- Added 'Open Issues' section on variable-rate links.

2. Introduction

The Van Jacobson congestion control algorithms [Jacobson88] [RFC2581] are used by the Internet transport protocols TCP [RFC0793] and SCTP [RFC2960]. The Jacobson algorithms are fundamental to stable and efficient Internet operation, and they have been highly successful over many orders of magnitude of Internet bandwidth and delay.

However, the Jacobson congestion control algorithms have begun to reach their limits. Gigabit-per-second file transfers, lossy wireless links, and high latency connections are all driving current TCP congestion control outside of its natural operating regime. The resulting performance problems are of great concern for important network applications.

The original Jacobson algorithm was a purely end-to-end solution, requiring no congestion-related state in routers. More recent modifications have backed off from this purity. Active queue management (AQM) in routers (e.g., RED) [RFC2309] improves performance by keeping queues small, while Explicit Congestion Notification (ECN) [RFC3168] passes one bit of congestion information back to senders. These measures do improve performance, but there is a limit to how much can be accomplished without more information from routers. The requirement of extreme scalability together with robustness has been a difficult hurdle to accelerating information flow.

This document concerns the Explicit Control Protocol (XCP) developed by Dina Katabi of MIT [KHR02]. XCP represents a significant advance

in Internet congestion control: it extracts congestion information directly from routers, without any per-flow state. XCP should be able to deliver the highest possible application performance over a broad range of network infrastructure, including extremely high speed and very high delay links that are not well served by the current control algorithms. XCP achieves fairness, maximum link utilization, and efficient use of bandwidth. XCP is novel in separating the efficiency and fairness policies of congestion control, enabling routers to put available capacity to work quickly while conservatively managing the allocation of capacity to flows. XCP is potentially applicable to any transport protocol, although initial testing has applied it to TCP in particular.

XCP's scalability is built upon the principle of carrying per-flow congestion state in packets. XCP packets carry a congestion header through which the sender requests a desired throughput. Routers make a fair per-flow bandwidth allocation without maintaining any per-flow state. This enables the sender to learn the bottleneck router's allocation to a particular flow in a single round trip.

The gains of XCP come with some pain. XCP is more difficult to deploy than other proposed Internet congestion control improvements, since it requires changes in the routers as well as in end systems. It will be necessary to develop and test XCP with real user traffic and in real environments, to gain experience with real router and host implementations and to collect data on performance. Providing specifications is an important step towards enabling experimentation which, in turn, will lead to deployment. XCP deployment issues will be addressed in more detail in a subsequent version of this document.

This document contains an initial specification of the protocol and algorithms used by XCP, as an experimental protocol. The XCP algorithms defined here are based upon Katabi's SIGCOMM paper [KHR02], her MIT thesis [Katabi03], and her ns simulation. However, this document includes algorithmic modifications and clarifications that have arisen from early experience with implementing and testing XCP at the USC Information Sciences Institute. (See http://www.isi.edu/isi-xcp for our project page.) This document is intended to provide a baseline for further engineering and testing of XCP.

This document is organized as follows. The remainder of Section 2

provides an overview of the XCP protocol, Section 3 discusses the format of the congestion header, Section 4 describes the functions occurring in the end-systems and routers, and Section 5 lists some unresolved issues.

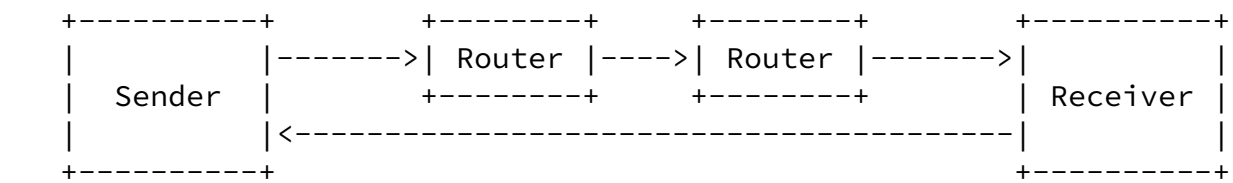
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.1. XCP Protocol Overview

The participants in the XCP protocol include sender hosts, receiver hosts, and intermediate nodes in which queuing occurs along the path from the sender to the receiver. The intermediate nodes are generally routers, but link-layer switches may also contain packet queues.

XCP supplies feedback from the network to the sender on the maximum rate (throughput) for injecting data into the network. XCP feedback is acquired through the use of a congestion header on each packet that is sent. Routers along the path may update the congestion header as it moves from the sender to the receiver. The receiver copies the network feedback into outbound packets of the same flow. An end-system may function as both a sender and a receiver in the case of a bidirectional flow.

The figure below illustrates four entities participating in XCP. The sender initializes the congestion header, two routers along the way may update it, and the receiver copies the feedback from the network into a returning packet in the same flow.



The congestion header contains four pieces of data:

- RTT: Set by the sender to its current estimate of the round-trip

time.

- X: Set by the sender to its current estimate of inter-packet time gap. This quantity is used in place of Throughput in earlier drafts of these specifications to avoid per-packet division in the router. See Section 4.1.1 for more about X.

- Delta_Throughput: Initialized to the amount which the sender would like to change (increase or decrease) its throughput, and updated by the routers along the path to be the network's allocated change in throughput. This value will be a negative number if an XCP-capable queue along the path wants the sender to slow down.

- Reverse_Feedback: When a data packet reaches the receiver, its Delta_Throughput value is returned to the sender in the Reverse_Feedback field of a congestion header of a returning packet (e.g., in an ACK packet).

An XCP-capable router calculates a fair capacity re-allocation for each packet. A flow only receives this re-allocation from a particular router if that router is the bottleneck for that flow. For XCP, a bottleneck router is defined to be a router that has insufficient capacity to accept a flow's current or desired throughput.

Based on current conditions, an XCP-capable router generates positive or negative feedback each time a packet arrives and compares it the packet's Delta_Throughput field. Delta_Throughput is reduced if the current value exceeds this calculated feedback allocation. Each XCP-

capable router along the path from sender to receiver performs this processing. A packet reaching the receiver therefore contains the minimal feedback allocation from the network, i.e., the capacity reallocation from the bottleneck router.

The receiver copies this value into the Reverse_Feedback field of a returning packet in the same flow (e.g., an ACK or DATA-ACK for TCP) and, in one round-trip, the sender learns the flow's per-packet

throughput allocation.

The sender uses the reverse feedback information to adjust its allowed sending rate. For the transport protocol TCP [RFC0793], for example, this may be accomplished by adjusting the congestion window, or cwnd, that limits the amount of unacknowledged data in the network. (Cwnd is defined for Van Jacobson congestion control in [RFC2581].)

Additionally, it is possible to use XCP's explicit notification of the bottleneck capacity allocation for other types of applications. For example, XCP may be implemented to support multimedia streams over DCCP [I-D.ietf-dccp-spec] or other transport protocols.

An XCP-capable router maintains two control algorithms on each output port: a congestion controller and a fairness controller. The congestion controller is responsible for making maximal use of the outbound link while at the same time draining any standing queues. The fairness controller is responsible for fairly allocating bandwidth to flows sharing the link. These two algorithms are executed only periodically, at an interval known as the "control interval". The algorithms defined below set this interval to the RTT averaged across all flows. Further work on choosing an appropriate value for the control interval may be required.

Each port-specific instance of XCP is independent of every other, and references to an "XCP router" should be considered an instance of XCP running on a particular output port.

> Actually, it is an oversimplification to say that congestion in routers only appears at output ports. Routers are complex devices which may experience resource contention in many forms and locations. Correctly expressing congestion which doesn't occur at the router output port is a topic for further study. Even so, it is important to correctly identify where the queue will build up in a router. The XCP algorithm will drain a standing queue; however it is necessary to measure that queue in order for correct operation. For more discussion of this issue see Section Section 5.5.

More context, analysis, and background can be found in [KHR02] and

[Katabi03].

3. The Congestion Header

The congestion control data required for XCP are placed in a new header which is called the Congestion Header.

3.1. Header placement

The Congestion Header is located between the IP and transport headers. This is consistent with the fact that XCP is neither hop-by-hop communication -- as in IP -- nor end-to-end communication -- as in TCP or UDP -- but is rather end-system-to-network communication. It should allow a router to "easily" locate the congestion header on a packet with no IP options.

Other choices were considered for header location. For example, making the Congestion Header a TCP option was suggested. This made sense as the congestion information is related to the transport protocol. However, it requires that routers be aware of the header format for every new transport protocol that might ever use XCP. This seemed like an unreasonable burden to place on the routers and would impede deployment of new transport protocols and/or XCP.

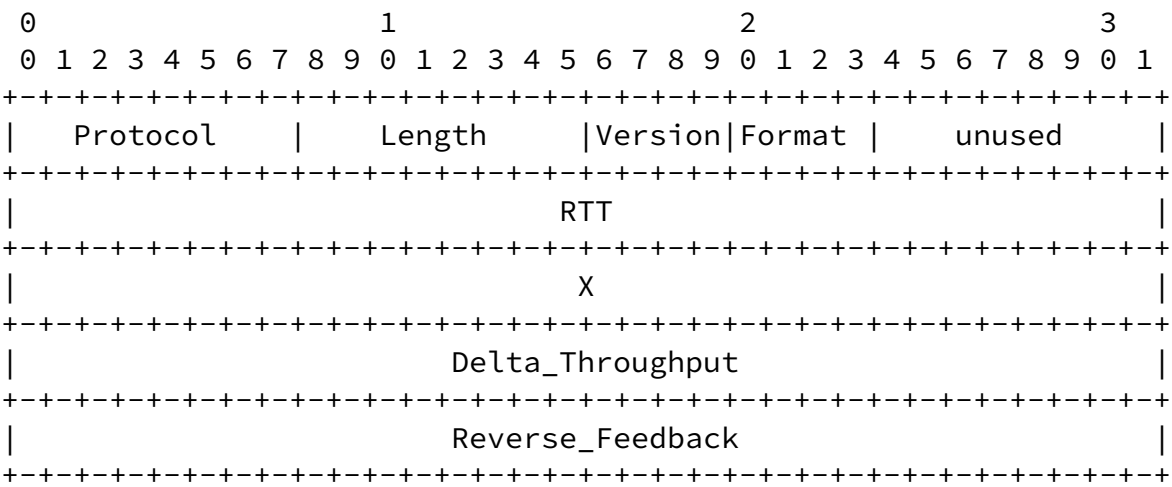
It has also been suggested that the Congestion Header be an IPv4-style option. While this proposal is transport protocol independent, it would generally force XCP packets to take the slow path on non-XCP routers along a path. This could severely impact performance.

The XCP protocol uses protocol number [TBD], assigned by IANA. IP packets containing XCP headers will use this protocol number in the IP header's Protocol field [RFC0791] to indicate to routers and end-systems that an XCP congestion header follows the IP header.

3.2. Congestion Header Formats

This section defines the XCP Congestion Header formats. This holds for IPv4; the corresponding header for IPv6 is a subject for further study.

XCP-capable IPv4 packets carry the following Congestion Header:



Protocol: 8 bits

This field indicates the next-level protocol used in the data portion of the packet. The values for various protocols are specified by IANA.

Length: 8 bits

This field indicates the length of the congestion header, measured in bytes. Length in this version of XCP will always be 20 bytes, or 0x14.

Version: 4 bits

This field indicates the version of XCP that is in use. The version of XCP described in this document corresponds to a value of 0x02. Future values will be assigned by IANA (http://www.iana.org). See note in the IANA Considerations section.

Format: 4 bits

This field contains a code to indicate the congestion header format. The current format codes are defined below.

Format	Code
Standard Format	0x1
Minimal Format	0x2

Table 1

Standard Format

The standard format includes the X, Delta_Throughput, and RTT fields shown above. This format is used by XCP in data packets flowing from sender to receiver.

Minimal Format

The X, Delta_Throughput, and RTT fields are unused and SHOULD be set to zero. A router MUST not perform any processing on a minimal format header. This format is intended for use in empty ACK packets, to return congestion information from receiver to sender.

Other formats may be defined in the future, to define different representation formats for the X, Delta_Throughput, and/or RTT fields, for example. The corresponding format values format values will be assigned by IANA. See IANA Considerations section below.

unused: 8 bits

This field is unused and MUST be set to zero in this version of

XCP.

RTT: 32bits

This field indicates the round-trip time measured by the sender, in fixed point format with 28 bits after the binary point, in seconds. Thus, the value of 1 corresponds to 2^(-28) of a second. This field is an unsigned integer.

The minimum value expressible in this field is 0s. A value of zero in the RTT field is legal and indicates that the sender either does not yet know the round-trip time, or operates at a coarse-grained timer granularity. The maximum value expressible in this field is 15.9999999963 seconds, in steps of 3.7ns.

X: 32 bits

This field indicates the inter-packet time of the flow as calculated by the sender, in fixed point format with 28 bits after the binary point, in seconds. This is the same format as used by the RTT field.

Delta_Throughput: 32 bits

This field indicates the desired or allocated change in throughput. It is set by the sender to indicate the amount by which the sender would like to adjust its throughput, and it may be subsequently reduced by routers along the path (See Section 4.2). It is measured in bytes per second and is a signed, 2's complement value.

The minimum throughput change expressible in this field is -17 Gbps. The maximum value expressible in this field is 17 Gbps, in steps of 8 bits per second.

Reverse_Feedback: 32bits

This field indicates the value of Delta_Throughput received by the data receiver. The receiver copies the field Delta_Throughput into the Reverse_Feedback field of the next outgoing packet in the

same flow. See Section 4.1.2.

3.3. IPsec issues

IPsec [RFC2401] must be slightly modified to accommodate use of XCP. The specifications for the IP Authenticated Header (AH) [RFC2402] and IP Encapsulating Security Payload (ESP) [RFC2406] state that the IPsec headers immediately follow the IP header. This would be a problem for XCP in that a) it would make the XCP headers harder to find by the routers, b) ESP encryption would make it impossible for routers along the path to read and write congestion header information and c) AH authentication would fail if any router along the path had modified a congestion header. Therefore, the XCP congestion header should immediately follow the IP header and precede any AH.

3.4. NAT, middlebox issues

Middleboxes that attempt to perform actions invisibly on flows must preserve the congestion header. Middleboxes that terminate the TCP connection should terminate the XCP connection. Middleboxes that insert queues into the forwarding path should participate in XCP.

3.5. MPLS/Tunneling Issues

When a flow enters an IP tunnel [RFC2003], IPsec ESP tunnel [RFC2406], or MPLS [RFC3031], network ingress point, the congestion header should be replicated on the "front" of the outer IP header. For example, when a packet enters an IP tunnel, the following transformation should occur:

```
              [IP2]  \_ outer header
      __-->   [XCP]  /
[IP1]/        [IP1]  \
[XCP] --->    [XCP]   |_ inner header
[TCP]         [TCP]   |
...           ...    /
```

Here the XCP header appended to the front of the outer header is copied from the inner header, with the appropriate change to the Protocol field to indicate that the next protocol is IP.

When the packet exits the tunnel, the congestion header, which may have been modified by routers along the tunneled path, is copied from the outer header into the inner header.

4. XCP Functions

XCP is concerned with the sender and receiver end-systems and the routers along the packet's path. This section describes the XCP-related algorithms to be implemented in each of these entities. The specific case of TCP as transport protocol is also described. The emphasis in this section is on explicit and detailed definition of the XCP algorithms. The theoretical derivation and analysis of the algorithms can found in [Katabi03].

4.1. End-System Functions

4.1.1. Sending Packets

The sender is responsible for maintaining five parameters, or their equivalent: (1) a desired throughput value, (2) a current estimate of the actual throughput, (3) the maximum throughput allowed by XCP, (4) a current estimate of inter-packet time, denoted X, and (5) a current estimate of the round-trip time

A sender may choose to use any reasonable value, i.e., any achievable value, for desired throughput. An application may supply this value via an API, or it might be the speed of the local interface.

When sending a packet, the sender fills in the fields of the congestion header as follows:

- The sender sets the RTT field to a scaled smoothed round-trip time estimate, or to zero if the round-trip time is not yet known.

- The sender sets the X field to the current inter-packet time estimate, or to zero if an estimate is not yet available. Packets carrying zero X field can receive negative feedback, but not positive. Since XCP requires only a single round trip for a flow to gain an estimate of RTT, this is expected to have negligible effect. The value of X may be estimated as the smoothed round-trip time estimate divided by the number of outstanding packets (or congestion window size in packets, for window-based protocols). Alternatively, it may be derived from the ratio of the packet size to the current throughput estimate. Using instantaneous RTT estimates in the calculation of X may yield better results than using the smoothed RTT, especially for senders with coarse-grained timers, i.e., timers with precision less than the RTT. This is a subject for further study. Also,see Section Section 4.1.3.3 for a discussion of RTT estimation.

- The sender calculates a desired change (typically, an increase) in throughput. This is normally the difference between the current estimated throughput and the desired throughput. However, if the sender does not have sufficient data to send at the current

allowed throughput, the desired change in throughput SHOULD be zero.

- The sender then divides the desired throughput change by the number of packets in one round-trip time, and puts the result in the Delta_Throughput field of the Congestion Header. This per-packet distribution of the throughput change is necessary because an XCP router does not maintain per-flow congestion state; it must treat each packet independently of others in the same flow. The number of packets in an RTT may be estimated by the product of the current throughput and the RTT, divided by the Maximum Segment Size (MSS).

  The Delta_Throughput (in bytes/second) can be calculated as:

$$\text{Delta\_Throughput} = \frac{\text{desired\_throughput} - \text{Throughput}}{\text{Throughput} * (\text{RTT}/\text{MSS})}$$

  where:
  desired_throughput is measured in bytes/second
  Throughput is measured in bytes/second
  RTT is measured in seconds
  MSS is measured in bytes

  However, Delta_Throughput should be set to zero if, for any reason, no additional capacity is needed, e.g., there is insufficient data to maintain Throughput for the next RTT, as discussed above.

- An issue for future consideration is how to treat the case when Delta_Throughput is calculated to be < 1 Bps. Since an integer representation is passed in the Congestion Header, the result will appear as zero. It would be possible to send a fraction of the packets in a round trip time with non-zero Delta_Throughput values.

- For TCP, the throughput estimate can be obtained by dividing the congestion window cwnd (in bytes) by RTT (in seconds), for example. Alternatively, it could be measured. The ratio cwnd/RTT differs from true throughput in two respects. First, cwnd doesn't account for header size. This may become significant should XCP

be applied to real-time flows that send large numbers of small packets, but it is probably not much worry for TCP flows that tend to use the largest possible packet size. Second, cwnd represents permission for the sender to transmit data. If the application doesn't use all of the available cwnd, the advertised throughput will be larger than the true throughput. This may result in an XCP router creating an unfair allocation of negative feedback to a flow.

4.1.2. Processing Feedback at the Receiver

An XCP receiver is responsible for copying the Delta_Throughput data it sees on arriving packets into the Reverse_Feedback field of outgoing packets. In TCP, outgoing packets would normally be ACK-only segments.

In some cases returning packets are sent less frequently than arriving packets, e.g., with delayed acknowledgments [RFC1122]. The receiver is responsible for calculating the sum of the arriving Delta_Throughput fields for placement in outgoing Reverse_Feedback fields.

4.1.2.1. Feedback from the Receiver

The receiver end-system returns XCP congestion feedback from the network to the sender, by copying the Delta_Throughput information from arriving packets into the Reverse_Feedback field of Congestion Headers in outgoing packets (possibly aggregating data as described in Section 4.1.2).

It is possible that even empty ACK packets may create or encounter congestion in the reverse direction. Although TCP implementations generally do not perform congestion-based pacing of empty ACK segments, some transport protocols (e.g., DCCP) may be. Such a transport protocol may choose to use XCP congestion control on the returning ACKs as well as on the data.

In the normal case of a unidirectional data flow with XCP applied only to that data flow, the feedback can be sent in a Minimal format Congestion Header, in which the RTT, X, and Delta_Throughput fields are set to zero.

4.1.3. Processing Feedback at the Sender

When packets arrive back to the sender carrying reverse feedback, the sender must adjust its sending rate accordingly.

As noted earlier, this throughput adjustment may be made in TCP by updating the sender's congestion window, cwnd. This should use the formula:

```
cwnd = max(cwnd + feedback * RTT, MSS)

where:
    cwnd     = current congestion window (bytes)
    feedback = Reverse_Feedback field from received packet,
               (bytes/sec, may be +/-)
    RTT      = Sender's current round-trip time estimate
               (seconds)
    MSS      = maximum segment size (bytes)
```

The value of cwnd has a minimum of MSS to avoid the "Silly Window Syndrome" [RFC0813].

4.1.3.1. Aging the Allowed Throughput

When a sending application does not send data fast enough to fully utilize the allowed throughput, XCP should reduce the allowed throughput as time passes, to avoid sudden bursts of data into the network if the application starts to send data later.

We present a slight modification of the algorithm for aging the allowed throughput below. It is based on Section 4.5 of [Katabi03]. Each RTT in which the sender sends with actual throughput which is less than the allowed throughput, the allowed throughput MUST be reduced by the following exponential averaging formula:

```
Allowed_Throughput = Allowed_Throughput*(1-p) +
                     Actual_Throughput * p

    where: p is a parameter controlling the speed of aging,
           ranged between 0 and 1.
```

Using p = 0.5 is suggested. Consideration of values of p or other algorithms is a research topic.

4.1.3.2. Response to Packet Loss

When the transport protocol is TCP, a packet drop or detection of an ECN notification [RFC3168] should trigger a transition to standard TCP congestion control behavior[RFC2581]. In other words, cwnd should be halved and Jacobson's fast retransmission/fast recovery, slow start, and congestion avoidance algorithms should be applied for the remainder of the connection or until the congestion event is known to have passed (see Section 5.1 for discussion of alternative

approaches to this issue. The assumption is that the packet drop reveals the presence of a congested non-XCP router in the path. Transitioning to standard TCP behavior is a conservative response.

Note also the following:

- The change in congestion control algorithm should be delayed until the three DUPACKs have arrived, according to the Fast Retransmission/Fast Recovery algorithm[RFC2581].

- Once the change to standard TCP congestion control has occurred, cwnd should be managed using the RFC2581 algorithm.

- The X field in outgoing packets should continue to reflect the current inter-packet time. This allows the XCP processes in the routers along the path to continue to monitor the flow's utilization.

- Further study is needed to determine whether it will be possible to return a connection to XCP congestion control, once it has transitioned to Van Jacobson mode.

- For transport protocols other than TCP, the response to a packet loss or ECN notification is a subject for further study.

4.1.3.3. RTT Estimates

Having a good estimate of the round trip time is more important in

XCP than in Van Jacobson congestion control. There is evidence that small errors in the RTT estimate can result in larger errors in the throughput and X estimates. The current cwnd divided by SRTT is only an approximation of the actual throughput. Likewise, SRTT divided by cwnd in packets is only an approximation of the highly variable inter-packet time, X. The RTT used in the ns-2 code in [KHR02] used a smoothed floating-point RTT estimator, rather than instantaneous measurements. Additional research is needed to develop recommendations for RTT estimation.

4.2. Router functions

The router calculations for XCP are divided into those that occur upon packet arrival, those that occur upon control interval timeout, those that occur upon packet departure, and the assessment of the

persistent queue, which uses a separate timer. The calculations are presented in the following sections as annotated pseudo-code.

4.2.1. Calculations Upon Packet Arrival

When a packet arrives at a router, several parameters used by XCP need to be updated. The steps are described in the following pseudo-code.

```
==========================================================
On packet arrival do:

1. input_traffic += Pkt_size

2. sum_x += X

3. if (Rtt < MAX_INTERVAL) then

4.    sum_xrtt += X * Rtt

5. else

6.    sum_xrtt += X * MAX_INTERVAL
==========================================================
```

Line 1: The variable input_traffic accumulates the volume of data

that have arrived during a control interval. When a packet arrives, the packet size is taken from the IP header and is added to the ongoing count.

Line 2: The variable sum_x is used in the control interval calculation (see equation 4.2 of [Katabi03]) and in capacity allocation. For each packet, values of X from the XCP header is accumulated. It is recommended that sum_x is stored in a 64-bit unsigned integer variable.

Lines 3 and 5: A test is performed to check whether the round trip time of the flow exceeds the maximum allowable control interval. If so, MAX_INTERVAL, the maximum allowable control interval, is used in the subsequent calculations. Too large a control interval will delay new flows from acquiring their fair allocation of capacity. See Section 4.2.4 for a discussion of the recommended value for MAX_INTERVAL.

Lines 4 and 6: As in Line 2, the variable sum_xrtt is used in the control interval calculation. It is recommended that it is stored in a 96-bit unsigned variable.

4.2.2. Calculations Upon Control Interval Timeout

When the control timer expires, several variables need to be updated as shown below.

Note that several calculations show divisions. These divisions should either be accomplished using floating-point arithmetic or integer arithmetic and appropriate scaling to avoid over- or under-flow.

========== On estimation-control timeout do:

7. avg_rtt = sum_xrtt / sum_x

```
 8. input_bw = input_traffic / ctl_interval

 9. F = a * (capacity - input_bw) - b * queue / avg_rtt

10. shuffled_traffic = shuffle_function(...)

11. residue_pos_fbk = shuffled_traffic + max(F,0)

12. residue_neg_fbk = shuffled_traffic + max(-F,0)

13. Cp = residue_pos_fbk / sum_x

14. Cn = residue_neg_fbk / input_traffic

15. input_traffic = 0

16. sum_x = 0

17. sum_xrtt = 0

18. ctl_interval = max(avg_rtt, MIN_INTERVAL)

19. timer.reschedule(ctl_interval)
=========================================================
```


Line 7: Update avg_rtt by taking the ratio of the two sums accumulated in the previous section. This value is used to determine the control interval (line 17).

Line 8: The average bandwidth of arriving traffic is calculated by dividing the bytes received in the previous control interval by the duration of the previous control interval.

Line 9: The aggregate feedback, F, is calculated. The variable 'capacity' is the ability of the outbound link to carry IP

packets, in bytes/second. The variable 'avg_rtt' was calculated in line 7. The variable 'queue' is the persistent queue and is defined in section Section 4.2.5. The values a and b are constant parameters. According to [Katabi03], the constant a may be any positive number such that a < (pi/4*sqrt(2)). A nominal value of 0.4 is recommended. The constant b is defined to be b = a^2*sqrt(2). (If the nominal value of a is used, the value for b would be 0.226.) Note that F may be positive or negative.

Line 10: This line establishes the amount of capacity that will be shuffled in the next control interval through the use of the shuffle_function. Shuffling takes a small amount of the available capacity and redistributes it by adding it to both the positive and negative feedback pools. This allows new flows to acquire capacity in a full loaded system.

The recommended shuffle_function is as follows:

$$shuffled\_traffic = \max(0, 0.1 * input\_bw - |F|)$$

The variable 'input_bw' is defined above in Line 8. Implementers may choose other functions. It is important to consider that more shuffled traffic decreases the time for new flows to acquire capacity and converge to fairness. However, too much shuffling may impede flows from acquiring their fair share of available capacity. (For example, consider a setup of N flows bottlenecked downstream from the given router and another flow, not limited as those, trying to acquire its fair share. In this case shuffling leads to under-utilization of the available bandwidth and impedes the unlimited flow.) Shuffled_traffic is always a positive value.

The objective of the feedback calculations is to obtain a per-packet

feedback allocation from the router. Lines 13 and 14 obtain factors in this calculation that, unfortunately, have no physical meaning. One might view them as per-flow capacity allocations that have some additional processing to prepare them for per-packet allocation. Note that, with the use of shuffled_traffic, a non-idle router will

always start a control interval with non-zero values for both Cn and Cp.

Line 11: The variable 'residue_pos_fbk' keeps track of the pool of available positive capacity a router has to allocate. It is initialized to the positive aggregate feedback.

Line 12: The variable 'residue_neg_fbk' keeps track of the pool of available negative capacity a router has to allocate. It is initialized to the negative aggregate feedback. This variable is always positive.

Line 13: This line calculates the positive feedback scale factor, Cp. The variables residue_pos_fbk, and sum_x are defined above.

Line 14: This line calculates the negative feedback scale factor, Cn. This is a positive value. The definitions for residue_neg_fbk, and input_traffic are given above.

Line 15-17: Reset various counters for the next control interval.

Line 18: Set the next control interval. The use of MIN_INTERVAL is important to establish a reasonable control interval when the router is idle.

Line 19: Set timer.

4.2.3. Calculations Upon Packet Departure

An XCP router processes each packet using the feedback parameters calculated above. As stated earlier, each packet indicates the current inter-packet time (X) and a throughput adjustment, Delta_Throughput. The router calculates a per-packet capacity change which will be compared to the Delta_Throughput field in the packet header. Using the AIMD rule, positive feedback is applied equally per-flow, while negative feedback is made proportional to each flow's capacity.

To accommodate high-speed routers, XCP uses a fixed-point numeric representation for the Congestion Header fields. This means that the per-packet calculations defined below result in residual error that is less than 1 Bps per packet. These errors accumulate across all the packets in a control interval, resulting in an inaccuracy in XCP's allocation of available bandwidth to flows. Further work is needed to understand whether this will be a significant problem and, if so, whether there is any solution short of using 64 bit precision or floating point.

Processing should be done according to the pseudo-code below.

```
=========================================================
On packet departure:

20. pos_fbk = Cp * X

21. neg_fbk = Cn * Pkt_size

22. feedback = pos_fbk - neg_fbk

23. if(Delta_Throughput > feedback) then

24.   Delta_Throughput = feedback

25. else

26.   neg_fbk = min(residue_neg_fbk, neg_fbk +
                          (feedback - Delta_Throughput))

27.   pos_fbk = Delta_Throughput + neg_fbk

28. residue_pos_fbk = max(0, residue_pos_fbk - pos_fbk)

29. residue_neg_fbk = max(0, residue_neg_fbk - neg_fbk)

30. if (residue_pos_fbk <= 0) then Cp = 0

31. if (residue_neg_fbk <= 0) then Cn = 0

=========================================================
```

Line 20: The contribution of positive feedback for the current packet is calculated using Cp, defined in line 13, and X (the flow's advertised inter-packet time) from the Congestion Header. Note that if Cp (and Cn in Line 21) is implemented as a floating point number, this calculation would be implemented by multiplying the Cp-mantissa by the value of X, then shifting the result by the

amount of the Cp-exponent.

Line 21: The contribution of negative feedback for the current packet is calculated using Cn, defined in line 14, and Pkt_size from the IP header. This value of neg_fbk is positive.

Line 22: The router's allocated feedback for the packet is the positive per-packet feedback minus the negative per-packet feedback. This value may be positive or negative.

Line 23-24: Line 23 tests whether the packet is requesting greater capacity increase (via the packet's Delta_Throughput field) than the router has allocated. If so, this means the the sender's desired throughput needs to be reduced to be the router's allocation. In line 24 the Delta_Throughput field in the packet header updated with the router feedback allocation.

Line 25: This branch is executed when the packet is requesting a smaller throughput increase than the router's allocation. In this branch, and the rest of this pseudo-code, the packet header is not updated and the remaining code is to correctly update the feedback pool variables.

Line 26: In this line, the packet's negative feedback contribution, neg_fbk, is set to be the smaller of two terms. The first term, residue_neg_fbk, is the pool of negative feedback, i.e., this drains the remaining negative feedback in the pool. The second term increases the nominal negative feedback from the router by the amount which the Delta_Throughput is less than net router allocation. This allows the router to capture feedback which is allocated by an upstream bottleneck.

Line 27: The positive allocation, pos_fbk, is adjusted to be the sum of Delta_Throughput and neg_fbk, from Line 26. This is required for the sum of pos_fbk and neg_fbk to equal Delta_Throughput.

Line 28-29: In these two lines, the feedback pools, residue_pos_fbk and residue_neg_fbk, are reduced by the values of pos_fbk and neg_fbk accordingly, but prevented from going negative.

Line 30-31: When a feedback pool becomes empty, set the scale factor to zero, i.e., stop handing out associated feedback.

4.2.4. The Control Interval

The capacity allocation algorithm in XCP router updates several parameters every Control Interval. The Control Interval is currently defined to be the average RTT of the flows passing through the router, i.e., avg_rtt calculated in Line 7 above. Other possible choices for the control interval are under study.

Notes on avg_rtt:

- In this document, the quantity 'avg_rtt' refers to the last calculated value. In other words, the avg_rtt calculated based on packets arriving in the previous control interval.

- The avg_rtt calculation should ignore packets with an RTT of zero in the header.

- avg_rtt MUST have a minimum value. This is to allow flows to acquire bandwidth from a previously idle router. The default minimum value, MIN_INTERVAL, should be max(5-10ms, propagation delay on attached link).

- avg_rtt MUST have a maximum value. The default maximum value, MAX_INTERVAL, should be max(0.5-1 sec, propagation delay on attached link).

4.2.5. Obtaining the Persistent Queue

In Section 4.2.2 the variable 'queue' contains the persistent queue over the control interval. This is intended to be the minimum standing queue over the queue estimation interval.

The following pseudo-code describes how to obtain the minimum persistent queue:


```
============================================================
On packet departure do:

32. min_queue = min(min_queue, inst_queue)

============================================================

When the queue-computation timer expires do:

33. queue = min_queue

34. min_queue = inst_queue

35. Tq = max(ALLOWED_QUEUE, (avg_rtt - inst_queue/capacity)/2)

36. queue_timer.reschedule(Tq)

============================================================
```

Line 32: The current instantaneous queue length is checked each time a packet departs compute the minimum queue size.

If avg_rtt is being used as the Control Interval, it MUST NOT be used as the interval for measuring the minimum persistent queue. Doing so can result in a feed-forward loop. For example, if a queue develops the average RTT will increase. If the avg_rtt increases, it takes longer to react to the growing queue and the queue gets larger, leading to instability.

Line 33: Upon expiration of the queue estimation timer, Tq, the variable queue, the persistent queue, is set to be the minimum queue occupancy over the last Tq.

Line 34: Upon expiration of the queue estimation timer, reset the running estimate of the minimum queue to be the current queue occupancy.

Line 35: The first term in the max function, ALLOWED_QUEUE, is the time to drain a standing queue that you are willing to tolerate. (A nominal value of 2ms worth of queuing is recommended but this may be tuned by implementers.) The second term is an estimate of the propagation delay. In other words the persistent queue is a queue that does not drain in a propagation delay. the division by 2 is a conservative factor to avoid overestimating the propagation delay.

Line 36: The queue computation timer is set.

5. Unresolved Issues

XCP is a work-in-progress. This section describes some known issues that need to be resolved.

5.1. XCP With Non-XCP Routers

Obviously, non-XCP routers will exist in networks before XCP becomes ubiquitously deployed and we expect other non-XCP systems to continue to be in the network indefinitely. Long term non-XCP network elements include any sort of link-level switches with queuing, e.g. ATM switches and sophisticated Ethernet switches. Even simple multiplexers are non-XCP queues with very little buffering.

Sources and the network care about these non-XCP elements because any one of them can be a site of network congestion, and if an XCP endpoint is bottlenecked at one of these non-XCP elements, no router feedback will inform the endpoint to slow down. If nothing is done, such an element will probably collapse under congestion.

Although exactly how XCP sources will operate in this environment is an open issue, a current promising direction is for endpoints to run a traditional end-to-end congestion detection algorithm in parallel with the XCP algorithm and switch over to using that algorithm for control when congestion is detected that XCP is not controlling. For example, an XCP source that detects 3 duplicate acknowledgments would fall back to TCP Reno behavior.

An endpoint that is limited by its end-to-end congestion algorithm would indicate so to XCP routers by setting a bit in the packet header. A router may process such packets differently than packets from endpoints that are being controlled by XCP. For example, the router might allocate end-to-end controlled packets less feedback or not reduce its feedback pools by the full amount when assigning feedback to those packets.

Though using its end-to-end algorithm to control its sending rate, an endpoint will also monitor the XCP feedback and if the source discovers that the XCP feedback would be more restrictive than the end-to-end control over a round trip time, the endpoint will revert to following XCP feedback. XCP feedback that is more restrictive over a round trip time is an indication that the endpoint's bottleneck is once again at an XCP router and the endpoint should take advantage of the more precise XCP information.

Evaluation of these algorithms is ongoing work

5.2. Variable Rate Links

As discussed in [Zhang05], XCP may perform poorly over shared links. When a link is shared, such as in CSMA ethernet or 802.11 wireless networks, a single queue's drain rate is often a function of the load in the shared medium. So, using a constant value for the variable 'capacity' in the routing control algorithm may not work well. For

correct operation, the XCP router's notion of capacity needs to reflect how the link capacity is shared.

5.3. XCP as a TCP PEP

In the Internet today TCP performance-enhancing proxies (PEPs) are sometimes used to improve application performance over certain networks. TCP PEPs, and the issues surrounding their use are described in [RFC3135]. A common mechanism used in TCP PEPs is to split a TCP connection into three parts where the first and last run TCP and a more aggressive transport protocol is run in the middle (across the path which generates poor TCP performance). This improves performance over the "problematic" portion of the path and does not require changing the protocol stacks on the end systems. For example, if a high-speed satellite link was used to connect a LAN to the Internet, a TCP PEP may be placed on either side of the satellite link.

It is not unusual today to find TCP PEPs which, to get high data rates, do not use congestion control at all. Of course, this limits the environments in which they can be used. However, XCP may be used in between two TCP PEPs to get high transfer rates and still respond to congestion in a correct and scalable way.

Work on using XCP as a TCP PEP is just beginning [Kapoor05]. Objectives for such a mechanism would be:

- preserve end-to-end TCP semantics as much as possible
- enable some form of discovery for one PEP to determine that another PEP was in the path
- allow for recovery should one half of a PEP pair fail or the route change so that one or both PEPs are not on the path
- enable aggregation of multiple flows between two PEPs.

A system which met the above objectives could also be used for incremental deployment of XCP. A network operator could deploy XCP-capable routers and use PEPs at the periphery of the network to convert from traditional TCP to XCP congestion control. This may

result in smaller queues and improved link utilization within the XCP network. (This may be of more interest to wireless network providers than to over-provisioned fiber backbones.)

5.4. Sharing resources between XCP and TCP

Katabi describes a system for sharing router bandwidth between XCP and TCP flows that is based on sharing the output capacity based on the average throughputs of XCP and TCP sources using the router.[Katabi03] Two queues are installed at the router and they are served with different weights by the forwarding engine. The algorithm is work-conserving; the forwarding engine is never idle when either queue has packets in it.

A TFRC-like algorithm estimates the average congestion window of TCP sources and the XCP algorithm estimates the average throughput of XCP sources. These averages are used to dynamically weight the time the processor spends on each queue. Initial experiments indicate that this system can provide fairness between the two flow classes (XCP/TCP).[Katabi03]

Open issues remain, however; for example there are questions about how well the TFRC-like algorithm can estimate TCP throughput with only access to local drop rates, convergence time of the weighting algorithm has never been explored, and no system for buffer allocation to complement the link capacity allocation has been put forward. These open issues are under study.

5.5. A Generalized Router Model

The XCP algorithm described here and in [Katabi03] manages congestion at a single point in a router, most likely an output queue. However, resource contention can occur at many points in a router. Input queues, backplanes, computational resources can 'congest' in addition to output buffers. There is a need to develop a general model and a variety of mechanisms to identify and manage resource contention throughout the router.

5.6. Host back-to-back operation

XCP hosts should be capable of back-to-back operation, i.e., with no router in the path. Nominally, this should not be a problem. A sender initializes delta_throughput to the desired value, no router modifies it and, thus, it is automatically granted. However, it has not yet been decided whether an XCP receiver should be capable of (or require) adjusting the delta_throughput to request flow control from the receiver to the sender.

At this point, XCP offers no mechanism for flow control. (Open question: Should it?) It is believed that running XCP on the output queue of a host would solve this problem. However, it isn't clear that the complexity is justified by the need to solve this situation.

6. Security Considerations

The presence of a header which may be read and written by entities not participating in the end-to-end communication opens some potential security vulnerabilities. This section describes them and tries to give enough context so that users can understand the risks.

Man-in-the-Middle Attacks

There is a man-in-the-middle attack where a malicious user can force a sender to stop sending by inserting negative feedback into flow. This is little different from a malicious user discarding packets belonging to a flow using VJ congestion control or setting ECN bits. One question worth investigating further is whether the XCP attack is harder to diagnose.

Covert Data Channels

IPsec needs to be modified, as discussed in Section 3.3, to allow routers to read the entire congestion header and write the delta_feedback field. This could become a covert data-channel, i.e., a way in which an end-system can make data viewable to observers in the network, on a compromised end-system.

Malicious Sources

The XCP algorithms rely on senders to advertise information about their current RTT and X and correctly respond to feedback delivered from the network. Naturally, the possibility occurs that a sender won't perform these functions correctly. Chapter 7 of [Katabi03] and [Katabi04] examine these issues.

A source which lies about its values of X and hence throughput

cannot affect the link utilization and, in the worst case, can unfairly acquire capacity. However, this is equivalent to a sender opening up multiple TCP flows. So, there is an incentive to lie about X. However, because X is explicitly stated in each packet header, it is a simpler matter to police it at the edge of the network than, say, for TCP.

A source which lies about its RTT can disrupt the router control algorithm, particularly when a large number of sources lie about their RTT and the router control interval is adaptive and uses the average RTT. However, there is little incentive to lie as it will

not affect the fair allocation of capacity and the liar will experience the same degradation as the non-lying flows. Lying about RTT should be considered a weak denial-of-service attack.

A flow may also ignore negative feedback from the router. Such a flow can obtain unfair throughput in a congested router. However, as with lying, the explicit nature of XCP makes it possible to verify that flows are responding to feedback. For example, a policing function in the path (presumeably near the edge so that the load is manageable and it can be expected to see packet flow in both directions) may inspect congestion headers for a flow in both directions. If the policer sees negative feedback heading towards a source and no reduction in throughput it may, e.g., punish the flow by severely restricting the throughput. Note that this can be applied on a probabilistic basis, sampling flows only occasionally.

7. IANA Considerations

XCP requires the assignment of an IP protocol number. Once this value has been assigned, the number may be inserted (by the RFC Editor) into Section 3.1 and this paragraph may be removed prior to publication.

8. Acknowledgements

The authors would like to acknowledge the many contributors who have assisted in this work. Bob Braden applied his usual sage guidance to the project and to the spec, in particular. Ted Faber wrote the initial implementation framework and provided much wisdom on kernel development and congestion control. John Wroclawski advised on project priorities and strategy. Eric Coe developed the initial implementation and testbed. Aman Kapoor performed supporting simulations and debugged kernel code. Padma Haldar ported ns-2 simulation code to ns-2 distribution. Jasmeet Bagga and Anuraag Mittal conducted simulations on various aspects of XCP performance. On the XCP mailing list, Tim Shepherd, Tom Henderson, and Matt Mathis made valuable contributions to the effort. To all the above go our sincere thanks.

9. References

[I-D.ietf-dccp-spec]
Kohler, E., "Datagram Congestion Control Protocol (DCCP)", draft-ietf-dccp-spec-04 (work in progress), July 2003.

[Jacobson88]
Jacobson, V., "Congestion Avoidance and Control", ACM Computer Communication Review Proceedings of the Sigcomm '88 Symposium, August 1988.

[KHR02] Katabi, D., Handley, M., and C. Rohr, "Internet Congestion Control for Future High Bandwidth-Delay Product Environments", ACM Computer Communication Review Proceedings of the Sigcomm '02 Symposium, August 2002.

[Kapoor05]
Kapoor, A., Falk, A., Faber, T., and Y. Pryadkin, "Achieving Faster Access to Satellite Link Bandwidth", IEEE 8th IEEE Global Internet Symposium, Miami, FL, March 2005, 2005.

[Katabi03]
Katabi, D., "Decoupling Congestion Control and Bandwidth Allocation Policy With Application to High Bandwidth-Delay Product Networks", MIT PhD. Thesis, March 2003.

[Katabi04]
Katabi, D., "XCP's Performance in the Presence of Malicious Flows", Second International Workshop on Protocols for Fast Long-Distance Networks, Presentation,

February 2004.

[Padhye98]
Padhye, J., Firoiu, V., Towsley, D., and J. Krusoe, "Modeling TCP throughput: A simple model and its empirical validation", ACM SIGCOMM '98 conference on Applications,technologies, architectures, and protocols for computer communication, 1998.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC0813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.

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

[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.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2402] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M.,

Zhang, L., and V. Paxson, "Stream Control Transmission

Protocol", RFC 2960, October 2000.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, January 2001.

[RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, June 2001.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[Zhang05] Zhang, Y. and T. Henderson, "An Implementation and Experimental Study of the eXplicit Control Protocol (XCP)", IEEE Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM 2005), pp. 1037-1048, Miami, Florida, Mar 2005., 2005.

Authors' Addresses

Aaron Falk
USC Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina Del Rey, CA 90292

Phone: 310-448-9327
Email: falk@isi.edu
URI: http://www.isi.edu/~falk

Yuri Pryadkin
USC Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina Del Rey, CA 90292

Phone: 310-448-8417
Email: yuri@isi.edu

Dina Katabi
Massachusetts Institute of Technology
200 Technology Square
Cambridge, MA 02139

Phone: 617-324-6027
Email: dk@mit.edu
URI: http://www.ana.lcs.mit.edu/dina/

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.