ï»¿

TCP Usage Guidance in the Internet of Things (IoT)

Abstract

   This document provides guidance on how to implement and use the
   Transmission Control Protocol (TCP) in Constrained-Node Networks
   (CNNs), which are a characteristic of the Internet of Things (IoT).
   Such environments require a lightweight TCP implementation and may
   not make use of optional functionality. This document explains a
   number of known and deployed techniques to simplify a TCP stack as
   well as corresponding trade-offs. The objective is to help embedded
   developers with decisions on which TCP features to use.

Status of This Memo

   This document is not an Internet Standards Track specification; it is
   published for informational purposes.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Not all documents
   approved by the IESG are candidates for any level of Internet
   Standard; see Section 2 of RFC 7841.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   https://www.rfc-editor.org/info/rfc9006.

Copyright Notice

Table of Contents

1. Introduction

   The Internet Protocol suite is being used for connecting Constrained-
   Node Networks (CNNs) to the Internet, enabling the so-called Internet
   of Things (IoT) [RFC7228]. In order to meet the requirements that
   stem from CNNs, the IETF has produced a suite of new protocols
   specifically designed for such environments (see, e.g., [RFC8352]).
   New IETF protocol stack components include the IPv6 over Low-Power
   Wireless Personal Area Networks (6LoWPANs) adaptation layer
   [RFC4944][RFC6282][RFC6775], the IPv6 Routing Protocol for Low-Power
   and Lossy Networks (RPL) [RFC6550], and the Constrained Application
   Protocol (CoAP) [RFC7252].

   As of this writing, the main transport-layer protocols in IP-based
   IoT scenarios are UDP and TCP. TCP has been criticized, often
   unfairly, as a protocol that is unsuitable for the IoT. It is true
   that some TCP features, such as relatively long header size,
   unsuitability for multicast, and always-confirmed data delivery, are
   not optimal for IoT scenarios. However, many typical claims on TCP
   unsuitability for IoT (e.g., a high complexity, connection-oriented
   approach incompatibility with radio duty-cycling and spurious
   congestion control activation in wireless links) are not valid, can
   be solved, or are also found in well-accepted IoT end-to-end
   reliability mechanisms (see a detailed analysis in [IntComp]).

   At the application layer, CoAP was developed over UDP [RFC7252].
   However, the integration of some CoAP deployments with existing
   infrastructure is being challenged by middleboxes such as firewalls,
   which may limit and even block UDP-based communications. This is the
   main reason why a CoAP over TCP specification has been developed
   [RFC8323].

   Other application-layer protocols not specifically designed for CNNs
   are also being considered for the IoT space. Some examples include
   HTTP/2 and even HTTP/1.1, both of which run over TCP by default
   [RFC7230] [RFC7540], and the Extensible Messaging and Presence
   Protocol (XMPP) [RFC6120]. TCP is also used by non-IETF application-
   layer protocols in the IoT space such as the Message Queuing
   Telemetry Transport (MQTT) [MQTT] and its lightweight variants.

   TCP is a sophisticated transport protocol that includes optional
   functionality (e.g., TCP options) that may improve performance in
   some environments. However, many optional TCP extensions require

complex logic inside the TCP stack and increase the code size and the
memory requirements. Many TCP extensions are not required for
interoperability with other standard-compliant TCP endpoints. Given
the limited resources on constrained devices, careful selection of
optional TCP features can make an implementation more lightweight.

This document provides guidance on how to implement and configure TCP
and guidance on how applications should use TCP in CNNs. The
overarching goal is to offer simple measures to allow for lightweight
TCP implementation and suitable operation in such environments. A
TCP implementation following the guidance in this document is
intended to be compatible with a TCP endpoint that is compliant to
the TCP standards, albeit possibly with a lower performance. This
implies that such a TCP client would always be able to connect with a
standard-compliant TCP server, and a corresponding TCP server would
always be able to connect with a standard-compliant TCP client.

This document assumes that the reader is familiar with TCP. A
comprehensive survey of the TCP standards can be found in RFC 7414
[RFC7414]. Similar guidance regarding the use of TCP in special
environments has been published before, e.g., for cellular wireless
networks [RFC3481].

2. Characteristics of CNNs Relevant for TCP

2.1. Network and Link Properties

CNNs are defined in [RFC7228] as networks whose characteristics are
influenced by being composed of a significant portion of constrained
nodes. The latter are characterized by significant limitations on
processing, memory, and energy resources, among others [RFC7228].
The first two dimensions pose constraints on the complexity and
memory footprint of the protocols that constrained nodes can support.
The latter requires techniques to save energy, such as radio duty-
cycling in wireless devices [RFC8352] and the minimization of the
number of messages transmitted/received (and their size).

[RFC7228] lists typical network constraints in CNNs, including low
achievable bitrate/throughput, high packet loss and high variability
of packet loss, highly asymmetric link characteristics, severe
penalties for using larger packets, limits on reachability over time,
etc. CNNs may use wireless or wired technologies (e.g., Power Line
Communication), and the transmission rates are typically low (e.g.,
below 1 Mbps).

For use of TCP, one challenge is that not all technologies in a CNN
may be aligned with typical Internet subnetwork design principles
[RFC3819]. For instance, constrained nodes often use physical- /
link-layer technologies that have been characterized as 'lossy',
i.e., exhibit a relatively high bit error rate. Dealing with
corruption loss is one of the open issues in the Internet [RFC6077].

2.2. Usage Scenarios

There are different deployment and usage scenarios for CNNs. Some
CNNs follow the star topology, whereby one or several hosts are
linked to a central device that acts as a router connecting the CNN
to the Internet. Alternatively, CNNs may also follow the multihop
topology [RFC6606].

In constrained environments, there can be different types of devices
[RFC7228]. For example, there can be devices with a single combined
send/receive buffer, a separate send and receive buffer, or a pool of
multiple send/receive buffers. In the latter case, it is possible
that buffers are also shared for other protocols.

One key use case for TCP in CNNs is a model where constrained devices
connect to unconstrained servers in the Internet. But it is also
possible that both TCP endpoints run on constrained devices. In the
first case, communication will possibly traverse a middlebox (e.g., a
firewall, NAT, etc.). Figure 1 illustrates such a scenario. Note

that the scenario is asymmetric, as the unconstrained device will
typically not suffer the severe constraints of the constrained
device. The unconstrained device is expected to be mains-powered,
have a high amount of memory and processing power, and be connected
to a resource-rich network.

Assuming that a majority of constrained devices will correspond to
sensor nodes, the amount of data traffic sent by constrained devices
(e.g., sensor node measurements) is expected to be higher than the
amount of data traffic in the opposite direction. Nevertheless,
constrained devices may receive requests (to which they may respond),
commands (for configuration purposes and for constrained devices
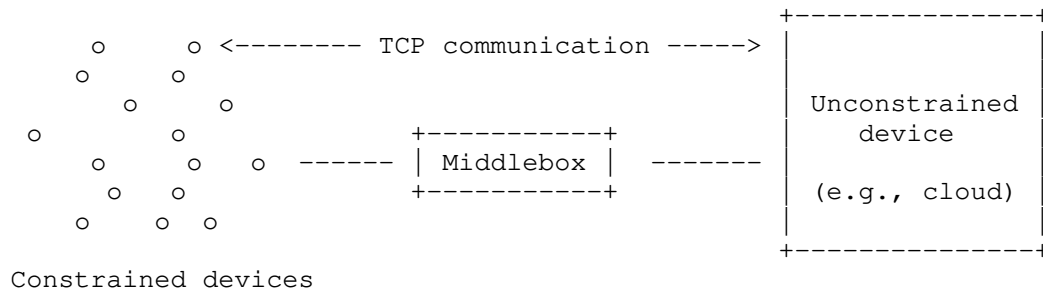including actuators), and relatively infrequent firmware/software
updates.

```
                                               +---------------+
        o       o <-------- TCP communication -----> |               |
      o     o                                    |               |
        o     o                                  |  Unconstrained |
     o       o                 +-----------+      |    device     |
       o     o   o  ------ | Middlebox |  ------- |               |
        o   o                 +-----------+      |  (e.g., cloud) |
       o     o   o                               |               |
                                                 +---------------+
   Constrained devices
```

           Figure 1: TCP Communication between a Constrained Device and an
                    Unconstrained Device, Traversing a Middlebox

2.3. Communication and Traffic Patterns

   IoT applications are characterized by a number of different
   communication patterns. The following non-comprehensive list
   explains some typical examples:

   Unidirectional transfers: An IoT device (e.g., a sensor) can
      (repeatedly) send updates to the other endpoint. There is not
      always a need for an application response back to the IoT device.

   Request-response patterns: An IoT device receiving a request from
      the other endpoint, which triggers a response from the IoT device.

   Bulk data transfers: A typical example for a long file transfer
      would be an IoT device firmware update.

   A typical communication pattern is that a constrained device
   communicates with an unconstrained device (cf. Figure 1). But it is
   also possible that constrained devices communicate amongst
   themselves.

3. TCP Implementation and Configuration in CNNs

   This section explains how a TCP stack can deal with typical
   constraints in CNN. The guidance in this section relates to the TCP
   implementation and its configuration.

3.1. Addressing Path Properties

3.1.1. Maximum Segment Size (MSS)

   Assuming that IPv6 is used, and for the sake of lightweight
   implementation and operation, unless applications require handling
   large data units (i.e., leading to an IPv6 datagram size greater than
   1280 bytes), it may be desirable to limit the IP datagram size to
   1280 bytes in order to avoid the need to support Path MTU Discovery
   [RFC8201]. In addition, an IP datagram size of 1280 bytes avoids
   incurring IPv6-layer fragmentation [RFC8900].

   An IPv6 datagram size exceeding 1280 bytes can be avoided by setting
   the TCP MSS to 1220 bytes or less. Note that it is already a

requirement for TCP implementations to consume payload space instead
of increasing datagram size when including IP or TCP options in an IP
packet to be sent [RFC6691]. Therefore, it is not required to
advertise an MSS smaller than 1220 bytes in order to accommodate TCP
options.

Note that setting the MTU to 1280 bytes is possible for link-layer
technologies in the CNN space, even if some of them are characterized
by a short data unit payload size, e.g., up to a few tens or hundreds
of bytes. For example, the maximum frame size in IEEE 802.15.4 is
127 bytes. 6LoWPAN defined an adaptation layer to support IPv6 over
IEEE 802.15.4 networks. The adaptation layer includes a
fragmentation mechanism, since IPv6 requires the layer below to
support an MTU of 1280 bytes [RFC8200], while IEEE 802.15.4 lacks
fragmentation mechanisms. 6LoWPAN defines an IEEE 802.15.4 link MTU
of 1280 bytes [RFC4944]. Other technologies, such as Bluetooth low
energy [RFC7668], ITU-T G.9959 [RFC7428], or Digital Enhanced
Cordless Telecommunications (DECT) Ultra Low Energy (ULE) [RFC8105],
also use 6LoWPAN-based adaptation layers in order to enable IPv6
support. These technologies do support link-layer fragmentation. By
exploiting this functionality, the adaptation layers that enable IPv6
over such technologies also define an MTU of 1280 bytes.

On the other hand, there exist technologies also used in the CNN
space, such as Master Slave (MS) / Token Passing (TP) [RFC8163],
Narrowband IoT (NB-IoT) [RFC8376], or IEEE 802.11ah [6LO-WLANAH],
that do not suffer the same degree of frame size limitations as the
technologies mentioned above. It is recommended that the MTU for MS/
TP be 1500 bytes [RFC8163]; the MTU in NB-IoT is 1600 bytes, and the
maximum frame payload size for IEEE 802.11ah is 7991 bytes.

Using a larger MSS (to a suitable extent) may be beneficial in some
scenarios, especially when transferring large payloads, as it reduces
the number of packets (and packet headers) required for a given
payload. However, the characteristics of the constrained network
need to be considered. In particular, in a lossy network where
unreliable fragment delivery is used, the amount of data that TCP
unnecessarily retransmits due to fragment loss increases (and
throughput decreases) quickly with the MSS. This happens because the
loss of a fragment leads to the loss of the whole fragmented packet
being transmitted. Unnecessary data retransmission is particularly
harmful in CNNs due to the resource constraints of such environments.
Note that, while the original 6LoWPAN fragmentation mechanism
[RFC4944] does not offer reliable fragment delivery, fragment
recovery functionality for 6LoWPAN or 6Lo environments has been
standardized [RFC8931].

3.1.2. Explicit Congestion Notification (ECN)

   ECN [RFC3168] allows a router to signal in the IP header of a packet
   that congestion is rising, for example, when a queue size reaches a
   certain threshold. An ECN-enabled TCP receiver will echo back the
   congestion signal to the TCP sender by setting a flag in its next TCP
   Acknowledgment (ACK). The sender triggers congestion control
   measures as if a packet loss had happened.

   RFC 8087 [RFC8087] outlines the principal gains in terms of increased
   throughput, reduced delay, and other benefits when ECN is used over a
   network path that includes equipment that supports Congestion
   Experienced (CE) marking. In the context of CNNs, a remarkable
   feature of ECN is that congestion can be signaled without incurring
   packet drops (which will lead to retransmissions and consumption of
   limited resources such as energy and bandwidth).

   ECN can further reduce packet losses since congestion control
   measures can be applied earlier [RFC2884]. Fewer lost packets
   implies that the number of retransmitted segments decreases, which is
   particularly beneficial in CNNs, where energy and bandwidth resources
   are typically limited. Also, it makes sense to try to avoid packet
   drops for transactional workloads with small data sizes, which are
   typical for CNNs. In such traffic patterns, it is more difficult and

often impossible to detect packet loss without retransmission
timeouts (e.g., as there may not be three duplicate ACKs). Any
retransmission timeout slows down the data transfer significantly.
In addition, if the constrained device uses power-saving techniques,
a retransmission timeout will incur a wake-up action, in contrast to
ACK clock-triggered sending. When the congestion window of a TCP
sender has a size of one segment and a TCP ACK with an ECN signal
(ECN-Echo (ECE) flag) arrives at the TCP sender, the TCP sender
resets the retransmit timer, and the sender will only be able to send
a new packet when the retransmit timer expires. Effectively, at that
moment, the TCP sender reduces its sending rate from 1 segment per
Round-Trip Time (RTT) to 1 segment per Retransmission Timeout (RTO)
and reduces the sending rate further on each ECN signal received in
subsequent TCP ACKs. Otherwise, if an ECN signal is not present in a
subsequent TCP ACK, the TCP sender resumes the normal ACK-clocked
transmission of segments [RFC3168].

ECN can be incrementally deployed in the Internet. Guidance on
configuration and usage of ECN is provided in RFC 7567 [RFC7567].
Given the benefits, more and more TCP stacks in the Internet support
ECN, and it makes sense to specifically leverage ECN in controlled
environments such as CNNs. As of this writing, there is ongoing work
to extend the types of TCP packets that are ECN capable, including
pure ACKs [TCPM-ECN]. Such a feature may further increase the
benefits of ECN in CNN environments. Note, however, that supporting
ECN increases implementation complexity.

3.1.3. Explicit Loss Notifications

There has been a significant body of research on solutions capable of
explicitly indicating whether a TCP segment loss is due to
corruption, in order to avoid activation of congestion control
mechanisms [ETEN] [RFC2757]. While such solutions may provide
significant improvement, they have not been widely deployed and
remain as experimental work. In fact, as of today, the IETF has not
standardized any such solution.

3.2. TCP Guidance for Single-MSS Stacks

This section discusses TCP stacks that allow transferring a single
MSS. More general guidance is provided in Section 3.3.

3.2.1. Single-MSS Stacks -- Benefits and Issues

A TCP stack can reduce the memory requirements by advertising a TCP
window size of 1 MSS and also transmit, at most, 1 MSS of
unacknowledged data. In that case, both congestion and flow control
implementation are quite simple. Such a small receive and send
window may be sufficient for simple message exchanges in the CNN
space. However, only using a window of 1 MSS can significantly
affect performance. A stop-and-wait operation results in low
throughput for transfers that exceed the length of 1 MSS, e.g., a
firmware download. Furthermore, a single-MSS solution relies solely
on timer-based loss recovery, therefore missing the performance gain
of Fast Retransmit and Fast Recovery (which requires a larger window
size; see Section 3.3.1).

If CoAP is used over TCP with the default setting for NSTART in RFC
7252 [RFC7252], a CoAP endpoint is not allowed to send a new message
to a destination until a response for the previous message sent to
that destination has been received. This is equivalent to an
application-layer window size of 1 data unit. For this use of CoAP,
a maximum TCP window of 1 MSS may be sufficient, as long as the CoAP
message size does not exceed 1 MSS. An exception in CoAP over TCP,
though, is the Capabilities and Settings Message (CSM) that must be
sent at the start of the TCP connection. The first application
message carrying user data is allowed to be sent immediately after
the CSM message. If the sum of the CSM size plus the application
message size exceeds the MSS, a sender using a single-MSS stack will
need to wait for the ACK confirming the CSM before sending the
application message.

3.2.2. TCP Options for Single-MSS Stacks

A TCP implementation needs to support, at a minimum, TCP options 2, 1, and 0. These are, respectively, the MSS option, the No-Operation option, and the End Of Option List marker [RFC0793]. None of these are a substantial burden to support. These options are sufficient for interoperability with a standard-compliant TCP endpoint, albeit many TCP stacks support additional options and can negotiate their use. A TCP implementation is permitted to silently ignore all other TCP options.

A TCP implementation for a constrained device that uses a single-MSS TCP receive or transmit window size may not benefit from supporting the following TCP options: Window Scale [RFC7323], TCP Timestamps [RFC7323], Selective Acknowledgment (SACK) [RFC2018], and SACK-Permitted [RFC2018]. Also, other TCP options may not be required on a constrained device with a very lightweight implementation. With regard to the Window Scale option, note that it is only useful if a window size greater than 64 kB is needed.

Note that a TCP sender can benefit from the TCP Timestamps option [RFC7323] in detecting spurious RTOs. The latter are quite likely to occur in CNN scenarios due to a number of reasons (e.g., route changes in a multihop scenario, link-layer retries, etc.). The header overhead incurred by the Timestamps option (of up to 12 bytes) needs to be taken into account.

3.2.3. Delayed Acknowledgments for Single-MSS Stacks

TCP Delayed Acknowledgments are meant to reduce the number of ACKs sent within a TCP connection, thus reducing network overhead, but they may increase the time until a sender may receive an ACK. In general, usefulness of Delayed ACKs depends heavily on the usage scenario (see Section 3.3.2). There can be interactions with single-MSS stacks.

When traffic is unidirectional, if the sender can send at most 1 MSS of data or the receiver advertises a receive window not greater than the MSS, Delayed ACKs may unnecessarily contribute delay (up to 500 ms) to the RTT [RFC5681], which limits the throughput and can increase data delivery time. Note that, in some cases, it may not be possible to disable Delayed ACKs. One known workaround is to split the data to be sent into two segments of smaller size. A standard-compliant TCP receiver may immediately acknowledge the second MSS of data, which can improve throughput. However, this "split hack" may not always work since a TCP receiver is required to acknowledge every second full-sized segment, but not two consecutive small segments. The overhead of sending two IP packets instead of one is another downside of the "split hack".

Similar issues may happen when the sender uses the Nagle algorithm, since the sender may need to wait for an unnecessarily Delayed ACK to send a new segment. Disabling the algorithm will not have impact if the sender can only handle stop-and-wait operation at the TCP level.

For request-response traffic, when the receiver uses Delayed ACKs, a response to a data message can piggyback an ACK, as long as the latter is sent before the Delayed ACK timer expires, thus avoiding unnecessary ACKs without payload. Disabling Delayed ACKs at the request sender allows an immediate ACK for the data segment carrying the response.

3.2.4. RTO Calculation for Single-MSS Stacks

The RTO calculation is one of the fundamental TCP algorithms [RFC6298]. There is a fundamental trade-off: a short, aggressive RTO behavior reduces wait time before retransmissions, but it also increases the probability of spurious timeouts. The latter leads to unnecessary waste of potentially scarce resources in CNNs such as energy and bandwidth. In contrast, a conservative timeout can result

in long error recovery times and, thus, needlessly delay data
delivery.

If a TCP sender uses a very small window size, and it cannot benefit
from Fast Retransmit and Fast Recovery or SACK, the RTO algorithm has
a large impact on performance. In that case, RTO algorithm tuning
may be considered, although careful assessment of possible drawbacks
is recommended [RFC8961].

As an example, adaptive RTO algorithms defined for CoAP over UDP have
been found to perform well in CNN scenarios [Commag] [CORE-FASOR].

3.3. General Recommendations for TCP in CNNs

This section summarizes some widely used techniques to improve TCP,
with a focus on their use in CNNs. The TCP extensions discussed here
are useful in a wide range of network scenarios, including CNNs.
This section is not comprehensive. A comprehensive survey of TCP
extensions is published in RFC 7414 [RFC7414].

3.3.1. Loss Recovery and Congestion/Flow Control

Devices that have enough memory to allow a larger (i.e., more than 3
MSS of data) TCP window size can leverage a more efficient loss
recovery than the timer-based approach used for a smaller TCP window
size (see Section 3.2.1) by using Fast Retransmit and Fast Recovery
[RFC5681], at the expense of slightly greater complexity and
Transmission Control Block (TCB) size. Assuming that Delayed ACKs
are used by the receiver, a window size of up to 5 MSS is required
for Fast Retransmit and Fast Recovery to work efficiently: in a given
TCP transmission of full-sized segments 1, 2, 3, 4, and 5, if segment
2 gets lost, and the ACK for segment 1 is held by the Delayed ACK
timer, then the sender should get an ACK for segment 1 when 3 arrives
and duplicate ACKs when segments 4, 5, and 6 arrive. It will
retransmit segment 2 when the third duplicate ACK arrives. In order
to have segments 2, 3, 4, 5, and 6 sent, the window has to be of at
least 5 MSS. With an MSS of 1220 bytes, a buffer of a size of 5 MSS
would require 6100 bytes.

The example in the previous paragraph did not use a further TCP
improvement such as Limited Transmit [RFC3042]. The latter may also
be useful for any transfer that has more than one segment in flight.
Small transfers tend to benefit more from Limited Transmit, because
they are more likely to not receive enough duplicate ACKs. Assuming
the example in the previous paragraph, Limited Transmit allows
sending 5 MSS with a congestion window (cwnd) of three segments, plus
two additional segments for the first two duplicate ACKs. With
Limited Transmit, even a cwnd of two segments allows sending 5 MSS,
at the expense of additional delay contributed by the Delayed ACK
timer for the ACK that confirms segment 1.

When a multiple-segment window is used, the receiver will need to
manage the reception of possible out-of-order received segments,
requiring sufficient buffer space. Note that even when a window of 1
MSS is used, out-of-order arrival should also be managed, as the
sender may send multiple sub-MSS packets that fit in the window. (On
the other hand, the receiver is free to simply drop out-of-order
segments, thus forcing retransmissions.)

3.3.1.1. Selective Acknowledgments (SACKs)

If a device with less severe memory and processing constraints can
afford advertising a TCP window size of several MSSs, it makes sense
to support the SACK option to improve performance. SACK allows a
data receiver to inform the data sender of non-contiguous data blocks
received, thus a sender (having previously sent the SACK-Permitted
option) can avoid performing unnecessary retransmissions, saving
energy and bandwidth, as well as reducing latency. In addition, SACK
often allows for faster loss recovery when there is more than one
lost segment in a window of data, since SACK recovery may complete
with less RTTs. SACK is particularly useful for bulk data transfers.

A receiver supporting SACK will need to keep track of the data blocks that need to be received. The sender will also need to keep track of which data segments need to be resent after learning which data blocks are missing at the receiver. SACK adds 8*n+2 bytes to the TCP header, where n denotes the number of data blocks received, up to four blocks. For a low number of out-of-order segments, the header overhead penalty of SACK is compensated by avoiding unnecessary retransmissions. When the sender discovers the data blocks that have already been received, it needs to also store the necessary state to avoid unnecessary retransmission of data segments that have already been received.

3.3.2. Delayed Acknowledgments

For certain traffic patterns, Delayed ACKs may have a detrimental effect, as already noted in Section 3.2.3. Advanced TCP stacks may use heuristics to determine the maximum delay for an ACK. For CNNs, the recommendation depends on the expected communication patterns.

When traffic over a CNN is expected mostly to be unidirectional messages with a size typically up to 1 MSS, and the time between two consecutive message transmissions is greater than the Delayed ACK timeout, it may make sense to use a smaller timeout or disable Delayed ACKs at the receiver. This avoids incurring additional delay, as well as the energy consumption of the sender (which might, e.g., keep its radio interface in receive mode) during that time. Note that disabling Delayed ACKs may only be possible if the peer device is administered by the same entity managing the constrained device. For request-response traffic, enabling Delayed ACKs is recommended at the server end, in order to allow combining a response with the ACK into a single segment, thus increasing efficiency. In addition, if a client issues requests infrequently, disabling Delayed ACKs at the client allows an immediate ACK for the data segment carrying the response.

In contrast, Delayed ACKs allow for a reduced number of ACKs in bulk transfer types of traffic, e.g., for firmware/software updates or for transferring larger data units containing a batch of sensor readings.

Note that, in many scenarios, the peer that a constrained device communicates with will be a general purpose system that communicates with both constrained and unconstrained devices. Since Delayed ACKs are often configured through system-wide parameters, the behavior of Delayed ACKs at the peer will be the same regardless of the nature of the endpoints it talks to. Such a peer will typically have Delayed ACKs enabled.

3.3.3. Initial Window

[RFC5681] specifies a TCP Initial Window (IW) of roughly 4 kB. Subsequently, RFC 6928 [RFC6928] defines an experimental new value for the IW, which in practice will result in an IW of 10 MSS. Nowadays, the latter is used in many TCP implementations.

Note that a 10-MSS IW was recommended for resource-rich environments (e.g., broadband environments), which are significantly different from CNNs. In CNNs, many application-layer data units are relatively small (e.g., below 1 MSS). However, larger objects (e.g., large files containing sensor readings, firmware updates, etc.) may also need to be transferred in CNNs. If such a large object is transferred in CNNs, with an IW setting of 10 MSS, there is significant buffer overflow risk, since many CNN devices support network or radio buffers of a size smaller than 10 MSS. In order to avoid such a problem, the IW needs to be carefully set in CNNs, based on device and network resource constraints. In many cases, a safe IW setting will be smaller than 10 MSS.

4. TCP Usage Recommendations in CNNs

This section discusses how TCP can be used by applications that are developed for CNN scenarios. These remarks are by and large

independent of how TCP is exactly implemented.

4.1. TCP Connection Initiation

In the scenario of a constrained device to an unconstrained device illustrated above, a TCP connection is typically initiated by the constrained device, in order for the device to support possible sleep periods to save energy.

4.2. Number of Concurrent Connections

TCP endpoints with a small amount of memory may only support a small number of connections. Each TCP connection requires storing a number of variables in the TCB. Depending on the internal TCP implementation, each connection may result in further memory overhead, and connections may compete for scarce resources (e.g., further memory overhead for send and receive buffers, etc.).

A careful application design may try to keep the number of concurrent connections as small as possible. A client can, for instance, limit the number of simultaneous open connections that it maintains to a given server. Multiple connections could, for instance, be used to avoid the "head-of-line blocking" problem in an application transfer. However, in addition to consuming resources, using multiple connections can also cause undesirable side effects in congested networks. For example, the HTTP/1.1 specification encourages clients to be conservative when opening multiple connections [RFC7230]. Furthermore, each new connection will start with a three-way handshake, therefore increasing message overhead.

Being conservative when opening multiple TCP connections is of particular importance in Constrained-Node Networks.

4.3. TCP Connection Lifetime

In order to minimize message overhead, it makes sense to keep a TCP connection open as long as the two TCP endpoints have more data to send. If applications exchange data rather infrequently, i.e., if TCP connections would stay idle for a long time, the idle time can result in problems. For instance, certain middleboxes such as firewalls or NAT devices are known to delete state records after an inactivity interval. RFC 5382 [RFC5382] specifies a minimum value for such an interval of 124 minutes. Measurement studies have reported that TCP NAT binding timeouts are highly variable across devices, with the median being around 60 minutes, the shortest timeout being around 2 minutes, and more than 50% of the devices with a timeout shorter than the aforementioned minimum timeout of 124 minutes [HomeGateway]. The timeout duration used by a middlebox implementation may not be known to the TCP endpoints.

In CNNs, such middleboxes may, e.g., be present at the boundary between the CNN and other networks. If the middlebox can be optimized for CNN use cases, it makes sense to increase the initial value for filter state inactivity timers to avoid problems with idle connections. Apart from that, this problem can be dealt with by different connection-handling strategies, each having pros and cons.

One approach for infrequent data transfer is to use short-lived TCP connections. Instead of trying to maintain a TCP connection for a long time, it is possible that short-lived connections can be opened between two endpoints, which are closed if no more data needs to be exchanged. For use cases that can cope with the additional messages and the latency resulting from starting new connections, it is recommended to use a sequence of short-lived connections instead of maintaining a single long-lived connection.

The message and latency overhead that stems from using a sequence of short-lived connections could be reduced by TCP Fast Open (TFO) [RFC7413], which is an experimental TCP extension, at the expense of increased implementation complexity and increased TCB size. TFO allows data to be carried in SYN (and SYN-ACK) segments and to be

consumed immediately by the receiving endpoint. This reduces the
message and latency overhead compared to the traditional three-way
handshake to establish a TCP connection. For security reasons, the
connection initiator has to request a TFO cookie from the other
endpoint. The cookie, with a size of 4 or 16 bytes, is then included
in SYN packets of subsequent connections. The cookie needs to be
refreshed (and obtained by the client) after a certain amount of
time. While a given cookie is used for multiple connections between
the same two endpoints, the latter may become vulnerable to privacy
threats. In addition, a valid cookie may be stolen from a
compromised host and may be used to perform SYN flood attacks, as
well as amplified reflection attacks to victim hosts (see Section 5
of [RFC7413]). Nevertheless, TFO is more efficient than frequently
opening new TCP connections with the traditional three-way handshake,
as long as the cookie can be reused in subsequent connections.
However, as stated in [RFC7413], TFO deviates from the standard TCP
semantics, since the data in the SYN could be replayed to an
application in some rare circumstances. Applications should not use
TFO unless they can tolerate this issue, e.g., by using TLS
[RFC7413]. A comprehensive discussion on TFO can be found in RFC
7413 [RFC7413].

Another approach is to use long-lived TCP connections with
application-layer heartbeat messages. Various application protocols
support such heartbeat messages (e.g., CoAP over TCP [RFC8323]).
Periodic application-layer heartbeats can prevent early filter state
record deletion in middleboxes. If the TCP binding timeout for a
middlebox to be traversed by a given connection is known, middlebox
filter state deletion will be avoided if the heartbeat period is
lower than the middlebox TCP binding timeout. Otherwise, the
implementer needs to take into account that middlebox TCP binding
timeouts fall in a wide range of possible values [HomeGateway], and
it may be hard to find a proper heartbeat period for application-
layer heartbeat messages.

One specific advantage of heartbeat messages is that they also allow
liveness checks at the application level. In general, it makes sense
to realize liveness checks at the highest protocol layer possible
that is meaningful to the application, in order to maximize the depth
of the liveness check. In addition, timely detection of a dead peer
may allow savings in terms of TCB memory use. However, the
transmission of heartbeat messages consumes resources. This aspect
needs to be assessed carefully, considering the characteristics of
each specific CNN.

A TCP implementation may also be able to send "keep-alive" segments
to test a TCP connection. According to [RFC1122], keep-alives are an
optional TCP mechanism that is turned off by default, i.e., an
application must explicitly enable it for a TCP connection. The
interval between keep-alive messages must be configurable, and it
must default to no less than two hours. With this large timeout, TCP
keep-alive messages might not always be useful to avoid deletion of
filter state records in some middleboxes. However, sending TCP keep-
alive probes more frequently risks draining power on energy-
constrained devices.

5. Security Considerations

   Best current practices for securing TCP and TCP-based communication
   also applies to CNN. As an example, use of TLS [RFC8446] is strongly
   recommended if it is applicable. However, note that TLS protects
   only the contents of the data segments.

   There are TCP options that can actually protect the transport layer.
   One example is the TCP Authentication Option (TCP-AO) [RFC5925].
   However, this option adds overhead and complexity. TCP-AO typically
   has a size of 16-20 bytes. An implementer needs to asses the trade-
   off between security and performance when using TCP-AO, considering
   the characteristics (in terms of energy, bandwidth, and computational
   power) of the environment where TCP will be used.

For the mechanisms discussed in this document, the corresponding
considerations apply. For instance, if TFO is used, the security
considerations of RFC 7413 [RFC7413] apply.

Constrained devices are expected to support smaller TCP window sizes
than less-limited devices. In such conditions, segment
retransmission triggered by RTO expiration is expected to be
relatively frequent, due to lack of (enough) duplicate ACKs,
especially when a constrained device uses a single-MSS
implementation. For this reason, constrained devices running TCP may
appear as particularly appealing victims of the so-called "shrew"
Denial-of-Service (DoS) attack [SHREW], whereby one or more sources
generate a packet spike targeted to coincide with consecutive RTO-
expiration-triggered retry attempts of a victim node. Note that the
attack may be performed by Internet-connected devices, including
constrained devices in the same CNN as the victim, as well as remote
ones. Mitigation techniques include RTO randomization and attack
blocking by routers able to detect shrew attacks based on their
traffic pattern.

6. IANA Considerations

   This document has no IANA actions.

7. References

7.1. Normative References

   [RFC0793] Postel, J., "Transmission Control Protocol", STD 7,
              RFC 793, DOI 10.17487/RFC0793, September 1981,
              <https://www.rfc-editor.org/info/rfc793>.

   [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -
              Communication Layers", STD 3, RFC 1122,
              DOI 10.17487/RFC1122, October 1989,
              <https://www.rfc-editor.org/info/rfc1122>.

   [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP
              Selective Acknowledgment Options", RFC 2018,
              DOI 10.17487/RFC2018, October 1996,
              <https://www.rfc-editor.org/info/rfc2018>.

   [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing
              TCP's Loss Recovery Using Limited Transmit", RFC 3042,
              DOI 10.17487/RFC3042, January 2001,
              <https://www.rfc-editor.org/info/rfc3042>.

   [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition
              of Explicit Congestion Notification (ECN) to IP",
              RFC 3168, DOI 10.17487/RFC3168, September 2001,
              <https://www.rfc-editor.org/info/rfc3168>.

   [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion
              Control", RFC 5681, DOI 10.17487/RFC5681, September 2009,
              <https://www.rfc-editor.org/info/rfc5681>.

   [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent,
              "Computing TCP's Retransmission Timer", RFC 6298,
              DOI 10.17487/RFC6298, June 2011,
              <https://www.rfc-editor.org/info/rfc6298>.

   [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)",
              RFC 6691, DOI 10.17487/RFC6691, July 2012,
              <https://www.rfc-editor.org/info/rfc6691>.

   [RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis,
              "Increasing TCP's Initial Window", RFC 6928,
              DOI 10.17487/RFC6928, April 2013,
              <https://www.rfc-editor.org/info/rfc6928>.

   [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for

                    Constrained-Node Networks", RFC 7228,
                    DOI 10.17487/RFC7228, May 2014,
                    <https://www.rfc-editor.org/info/rfc7228>.

     [RFC7323]   Borman, D., Braden, B., Jacobson, V., and R.
                    Scheffenegger, Ed., "TCP Extensions for High Performance",
                    RFC 7323, DOI 10.17487/RFC7323, September 2014,
                    <https://www.rfc-editor.org/info/rfc7323>.

     [RFC7413]   Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP
                    Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014,
                    <https://www.rfc-editor.org/info/rfc7413>.

     [RFC7567]   Baker, F., Ed. and G. Fairhurst, Ed., "IETF
                    Recommendations Regarding Active Queue Management",
                    BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015,
                    <https://www.rfc-editor.org/info/rfc7567>.

     [RFC8200]   Deering, S. and R. Hinden, "Internet Protocol, Version 6
                    (IPv6) Specification", STD 86, RFC 8200,
                    DOI 10.17487/RFC8200, July 2017,
                    <https://www.rfc-editor.org/info/rfc8200>.

7.2.   Informative References

     [6LO-WLANAH]
                    Del Carpio Vega, L., Robles, M., and R. Morabito, "IPv6
                    over 802.11ah", Work in Progress, Internet-Draft, draft-
                    delcarpio-6lo-wlanah-01, 19 October 2015,
                    <https://tools.ietf.org/html/draft-delcarpio-6lo-wlanah-
                    01>.

     [Commag]   Betzler, A., Gomez, C., Demirkol, I., and J. Paradells,
                    "CoAP Congestion Control for the Internet of Things", IEEE
                    Communications Magazine, Vol. 54, Issue 7, pp. 154-160,
                    DOI 10.1109/MCOM.2016.7509394, July 2016,
                    <https://doi.org/10.1109/MCOM.2016.7509394>.

     [CORE-FASOR]
                    Jarvinen, I., Kojo, M., Raitahila, I., and Z. Cao, "Fast-
                    Slow Retransmission Timeout and Congestion Control
                    Algorithm for CoAP", Work in Progress, Internet-Draft,
                    draft-ietf-core-fasor-01, 19 October 2020,
                    <https://tools.ietf.org/html/draft-ietf-core-fasor-01>.

     [Dunk]   Dunkels, A., "Full TCP/IP for 8-Bit Architectures",
                    MobiSys '03, pp. 85-98, DOI 10.1145/1066116.106611, May
                    2003, <https://doi.org/10.1145/1066116.106611>.

     [ETEN]   Krishnan, R., Sterbenz, J., Eddy, W., and C. Partridge,
                    "Explicit transport error notification (ETEN) for error-
                    prone wireless and satellite networks", Computer Networks,
                    DOI 10.1016/j.comnet.2004.06.012, June 2004,
                    <https://doi.org/10.1016/j.comnet.2004.06.012>.

     [GNRC]   Lenders, M., Kietzmann, P., Hahm, O., Petersen, H.,
                    GÃ¼ndoÄ\237a, C., Baccelli, E., Schleiser, K., Schmidt, T., and
                    M. WÃ¤hlisch, "Connecting the World of Embedded Mobiles:
                    The RIOT Approach to Ubiquitous Networking for the IoT",
                    arXiv:1801.02833v1 [cs.NI], January 2018.

     [HomeGateway]
                    Haetoenen, S., Nyrhinen, A., Eggert, L., Strowes, S.,
                    Sarolahti, P., and M. Kojo, "An Experimental Study of Home
                    Gateway Characteristics", Proceedings of the 10th ACM
                    SIGCOMM conference on Internet measurement, pp. 260-266,
                    DOI 10.1145/1879141.1879174, November 2010,
                    <https://doi.org/10.1145/1879141.1879174>.

     [IntComp]   Gomez, C., Arcia-Moret, A., and J. Crowcroft, "TCP in the
                    Internet of Things: from Ostracism to Prominence", IEEE

```
             Internet Computing, Vol. 22, Issue 1, pp. 29-41,
             DOI 10.1109/MIC.2018.112102200, January 2018,
             <https://doi.org/10.1109/MIC.2018.112102200>.

   [MQTT]     ISO/IEC, "Information technology -- Message Queuing
             Telemetry Transport (MQTT) v3.1.1", ISO/IEC 20922:2016,
             June 2016.

   [RFC2757]  Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N.
             Vaidya, "Long Thin Networks", RFC 2757,
             DOI 10.17487/RFC2757, January 2000,
             <https://www.rfc-editor.org/info/rfc2757>.

   [RFC2884]  Hadi Salim, J. and U. Ahmed, "Performance Evaluation of
             Explicit Congestion Notification (ECN) in IP Networks",
             RFC 2884, DOI 10.17487/RFC2884, July 2000,
             <https://www.rfc-editor.org/info/rfc2884>.

   [RFC3481]  Inamura, H., Ed., Montenegro, G., Ed., Ludwig, R., Gurtov,
             A., and F. Khafizov, "TCP over Second (2.5G) and Third
             (3G) Generation Wireless Networks", BCP 71, RFC 3481,
             DOI 10.17487/RFC3481, February 2003,
             <https://www.rfc-editor.org/info/rfc3481>.

   [RFC3819]  Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D.,
             Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L.
             Wood, "Advice for Internet Subnetwork Designers", BCP 89,
             RFC 3819, DOI 10.17487/RFC3819, July 2004,
             <https://www.rfc-editor.org/info/rfc3819>.

   [RFC4944]  Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler,
             "Transmission of IPv6 Packets over IEEE 802.15.4
             Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007,
             <https://www.rfc-editor.org/info/rfc4944>.

   [RFC5382]  Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P.
             Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142,
             RFC 5382, DOI 10.17487/RFC5382, October 2008,
             <https://www.rfc-editor.org/info/rfc5382>.

   [RFC5925]  Touch, J., Mankin, A., and R. Bonica, "The TCP
             Authentication Option", RFC 5925, DOI 10.17487/RFC5925,
             June 2010, <https://www.rfc-editor.org/info/rfc5925>.

   [RFC6077]  Papadimitriou, D., Ed., Welzl, M., Scharf, M., and B.
             Briscoe, "Open Research Issues in Internet Congestion
             Control", RFC 6077, DOI 10.17487/RFC6077, February 2011,
             <https://www.rfc-editor.org/info/rfc6077>.

   [RFC6120]  Saint-Andre, P., "Extensible Messaging and Presence
             Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120,
             March 2011, <https://www.rfc-editor.org/info/rfc6120>.

   [RFC6282]  Hui, J., Ed. and P. Thubert, "Compression Format for IPv6
             Datagrams over IEEE 802.15.4-Based Networks", RFC 6282,
             DOI 10.17487/RFC6282, September 2011,
             <https://www.rfc-editor.org/info/rfc6282>.

   [RFC6550]  Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J.,
             Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur,
             JP., and R. Alexander, "RPL: IPv6 Routing Protocol for
             Low-Power and Lossy Networks", RFC 6550,
             DOI 10.17487/RFC6550, March 2012,
             <https://www.rfc-editor.org/info/rfc6550>.

   [RFC6606]  Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem
             Statement and Requirements for IPv6 over Low-Power
             Wireless Personal Area Network (6LoWPAN) Routing",
             RFC 6606, DOI 10.17487/RFC6606, May 2012,
             <https://www.rfc-editor.org/info/rfc6606>.
```

[RFC6775]   Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C.
            Bormann, "Neighbor Discovery Optimization for IPv6 over
            Low-Power Wireless Personal Area Networks (6LoWPANs)",
            RFC 6775, DOI 10.17487/RFC6775, November 2012,
            <https://www.rfc-editor.org/info/rfc6775>.

[RFC7230]   Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
            Protocol (HTTP/1.1): Message Syntax and Routing",
            RFC 7230, DOI 10.17487/RFC7230, June 2014,
            <https://www.rfc-editor.org/info/rfc7230>.

[RFC7252]   Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
            Application Protocol (CoAP)", RFC 7252,
            DOI 10.17487/RFC7252, June 2014,
            <https://www.rfc-editor.org/info/rfc7252>.

[RFC7414]   Duke, M., Braden, R., Eddy, W., Blanton, E., and A.
            Zimmermann, "A Roadmap for Transmission Control Protocol
            (TCP) Specification Documents", RFC 7414,
            DOI 10.17487/RFC7414, February 2015,
            <https://www.rfc-editor.org/info/rfc7414>.

[RFC7428]   Brandt, A. and J. Buron, "Transmission of IPv6 Packets
            over ITU-T G.9959 Networks", RFC 7428,
            DOI 10.17487/RFC7428, February 2015,
            <https://www.rfc-editor.org/info/rfc7428>.

[RFC7540]   Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext
            Transfer Protocol Version 2 (HTTP/2)", RFC 7540,
            DOI 10.17487/RFC7540, May 2015,
            <https://www.rfc-editor.org/info/rfc7540>.

[RFC7668]   Nieminen, J., Savolainen, T., Isomaki, M., Patil, B.,
            Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low
            Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015,
            <https://www.rfc-editor.org/info/rfc7668>.

[RFC8087]   Fairhurst, G. and M. Welzl, "The Benefits of Using
            Explicit Congestion Notification (ECN)", RFC 8087,
            DOI 10.17487/RFC8087, March 2017,
            <https://www.rfc-editor.org/info/rfc8087>.

[RFC8105]   Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt,
            M., and D. Barthel, "Transmission of IPv6 Packets over
            Digital Enhanced Cordless Telecommunications (DECT) Ultra
            Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May
            2017, <https://www.rfc-editor.org/info/rfc8105>.

[RFC8163]   Lynn, K., Ed., Martocci, J., Neilson, C., and S.
            Donaldson, "Transmission of IPv6 over Master-Slave/Token-
            Passing (MS/TP) Networks", RFC 8163, DOI 10.17487/RFC8163,
            May 2017, <https://www.rfc-editor.org/info/rfc8163>.

[RFC8201]   McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed.,
            "Path MTU Discovery for IP version 6", STD 87, RFC 8201,
            DOI 10.17487/RFC8201, July 2017,
            <https://www.rfc-editor.org/info/rfc8201>.

[RFC8323]   Bormann, C., Lemay, S., Tschofenig, H., Hartke, K.,
            Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained
            Application Protocol) over TCP, TLS, and WebSockets",
            RFC 8323, DOI 10.17487/RFC8323, February 2018,
            <https://www.rfc-editor.org/info/rfc8323>.

[RFC8352]   Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, Ed.,
            "Energy-Efficient Features of Internet of Things
            Protocols", RFC 8352, DOI 10.17487/RFC8352, April 2018,
            <https://www.rfc-editor.org/info/rfc8352>.

[RFC8376]   Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN)
            Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018,

                    <https://www.rfc-editor.org/info/rfc8376>.

   [RFC8446]   Rescorla, E., "The Transport Layer Security (TLS) Protocol
               Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
               <https://www.rfc-editor.org/info/rfc8446>.

   [RFC8900]   Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O.,
               and F. Gont, "IP Fragmentation Considered Fragile",
               BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020,
               <https://www.rfc-editor.org/info/rfc8900>.

   [RFC8931]   Thubert, P., Ed., "IPv6 over Low-Power Wireless Personal
               Area Network (6LoWPAN) Selective Fragment Recovery",
               RFC 8931, DOI 10.17487/RFC8931, November 2020,
               <https://www.rfc-editor.org/info/rfc8931>.

   [RFC8961]   Allman, M., "Requirements for Time-Based Loss Detection",
               BCP 233, RFC 8961, DOI 10.17487/RFC8961, November 2020,
               <https://www.rfc-editor.org/info/rfc8961>.

   [RIOT]   Baccelli, E., GÃ¼ndoÄ\237a, C., Hahm, O., Kietzmann, P.,
               Lenders, M., Petersen, H., Schleiser, K., Schmidt, T., and
               M. WÃ¤hlisch, "RIOT: An Open Source Operating System for
               Low-End Embedded Devices in the IoT", IEEE Internet of
               Things Journal, Vol. 5, Issue 6,
               DOI 10.1109/JIOT.2018.2815038, March 2018,
               <https://doi.org/10.1109/JIOT.2018.2815038>.

   [SHREW]   Nyrhinen, A. and E. Knightly, "Low-Rate TCP-Targeted
               Denial of Service Attacks (The Shrew vs. the Mice and
               Elephants)", SIGCOMM'03, DOI 10.1145/863955.863966, August
               2003, <https://doi.org/10.1145/863955.863966>.

   [TCPM-ECN] Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit
               Congestion Notification (ECN) to TCP Control Packets",
               Work in Progress, Internet-Draft, draft-ietf-tcpm-
               generalized-ecn-07, 16 February 2021,
               <https://tools.ietf.org/html/draft-ietf-tcpm-generalized-
               ecn-07>.

Appendix A. TCP Implementations for Constrained Devices

   This section overviews the main features of TCP implementations for
   constrained devices. The survey is limited to open-source stacks
   with a small footprint. It is not meant to be all-encompassing. For
   more powerful embedded systems (e.g., with 32-bit processors), there
   are further stacks that comprehensively implement TCP. On the other
   hand, please be aware that this Annex is based on information
   available as of the writing.

A.1. uIP

   uIP is a TCP/IP stack, targeted for 8- and 16-bit microcontrollers,
   which pioneered TCP/IP implementations for constrained devices. uIP
   has been deployed with Contiki and the Arduino Ethernet shield. A
   code size of ~5 kB (which comprises checksumming, IPv4, ICMP, and
   TCP) has been reported for uIP [Dunk]. Later versions of uIP
   implement IPv6 as well.

   uIP uses the same global buffer for both incoming and outgoing
   traffic, which has a size of a single packet. In case of a
   retransmission, an application must be able to reproduce the same
   user data that had been transmitted. Multiple connections are
   supported but need to share the global buffer.

   The MSS is announced via the MSS option on connection establishment,
   and the receive window size (of 1 MSS) is not modified during a
   connection. Stop-and-wait operation is used for sending data. Among
   other optimizations, this allows for the avoidance of sliding window
   operations, which use 32-bit arithmetic extensively and are expensive
   on 8-bit CPUs.

Contiki uses the "split hack" technique (see Section 3.2.3) to avoid Delayed ACKs for senders using a single segment.

The code size of the TCP implementation in Contiki-NG has been measured to be 3.2 kB on CC2538DK, cross-compiling on Linux.

A.2. lwIP

lwIP is a TCP/IP stack, targeted for 8- and 16-bit microcontrollers. lwIP has a total code size of ~14 kB to ~22 kB (which comprises memory management, checksumming, network interfaces, IPv4, ICMP, and TCP) and a TCP code size of ~9 kB to ~14 kB [Dunk]. Both IPv4 and IPv6 are supported in lwIP since v2.0.0.

In contrast with uIP, lwIP decouples applications from the network stack. lwIP supports a TCP transmission window greater than a single segment, as well as the buffering of incoming and outgoing data. Other implemented mechanisms comprise slow start, congestion avoidance, fast retransmit, and fast recovery. SACK and Window Scale support has been recently added to lwIP.

A.3. RIOT

The RIOT TCP implementation (called "GNRC TCP") has been designed for Class 1 devices [RFC7228]. The main target platforms are 8- and 16-bit microcontrollers, with 32-bit platforms also supported. GNRC TCP offers a similar function set as uIP, but it provides and maintains an independent receive buffer for each connection. In contrast to uIP, retransmission is also handled by GNRC TCP. For simplicity, GNRC TCP uses a single-MSS implementation. The application programmer does not need to know anything about the TCP internals; therefore, GNRC TCP can be seen as a user-friendly uIP TCP implementation.

The MSS is set on connections establishment and cannot be changed during connection lifetime. GNRC TCP allows multiple connections in parallel, but each TCB must be allocated somewhere in the system. By default, there is only enough memory allocated for a single TCP connection, but it can be increased at compile time if the user needs multiple parallel connections.

The RIOT TCP implementation offers an optional Portable Operating System Interface (POSIX) socket wrapper that enables POSIX compliance, if needed.

Further details on RIOT and GNRC can be found in [RIOT] and [GNRC].

A.4. TinyOS

TinyOS was important as a platform for early constrained devices. TinyOS has an experimental TCP stack that uses a simple non-blocking library-based implementation of TCP, which provides a subset of the socket interface primitives. The application is responsible for buffering. The TCP library does not do any receive-side buffering. Instead, it will immediately dispatch new, in-order data to the application or otherwise drop the segment. A send buffer is provided by the application. Multiple TCP connections are possible. Recently, there has been little work on the stack.

A.5. FreeRTOS

FreeRTOS is a real-time operating system kernel for embedded devices that is supported by 16- and 32-bit microprocessors. Its TCP implementation is based on multiple-segment window size, although a "Tiny-TCP" option, which is a single-MSS variant, can be enabled. Delayed ACKs are supported, with a 20 ms Delayed ACK timer as a technique intended "to gain performance".

A.6. uC/OS

uC/OS is a real-time operating system kernel for embedded devices,
which is maintained by Micrium. uC/OS is intended for 8-, 16-, and
32-bit microprocessors. The uC/OS TCP implementation supports a
multiple-segment window size.

A.7. Summary

None of the implementations considered in this Annex support ECN or
TFO.

	uIP	lwIP orig	lwIP 2.1	RIOT	TinyOS	FreeRTOS	uC/OS
Code Size (kB)	<5	~9 to ~14	38	<7	N/A	<9.2	N/A
Memory	(a)	(T1)	(T4)	(T3)	N/A	(T2)	N/A
TCP Features							
Single- Segm.	Yes	No	No	Yes	No	No	No
Slow start	No	Yes	Yes	No	Yes	No	Yes
Fast rec/retx	No	Yes	Yes	No	Yes	No	Yes
Keep- alive	No	No	Yes	No	No	Yes	Yes
Win. Scale	No	No	Yes	No	No	Yes	No
TCP timest.	No	No	Yes	No	No	Yes	No
SACK	No	No	Yes	No	No	Yes	No
Del. ACKs	No	Yes	Yes	No	No	Yes	Yes
Socket	No	No	Optional	(I)	Subset	Yes	Yes
Concur. Conn.	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TLS supported	No	No	Yes	Yes	Yes	Yes	Yes

Table 1: Summary of TCP Features for Different Lightweight TCP
Implementations

Legend:

(T1):   TCP-only, on x86 and AVR platforms

(T2):   TCP-only, on ARM Cortex-M platform

(T3):   TCP-only, on ARM Cortex-M0+ platform (NOTE: RAM usage for the
        same platform is ~2.5 kB for one TCP connection plus ~1.2 kB
        for each additional connection)

(T4):   TCP-only, on CC2538DK, cross-compiling on Linux

(a):   Includes IP, ICMP, and TCP on x86 and AVR platforms. The

Authors' Addresses

   Carles Gomez
   UPC
   C/Esteve Terradas, 7
   08860 Castelldefels
   Spain

   Email: carlesgo@entel.upc.edu

   Jon Crowcroft
   University of Cambridge
   JJ Thomson Avenue
   Cambridge
   CB3 0FD
   United Kingdom

   Email: jon.crowcroft@cl.cam.ac.uk

   Michael Scharf
   Hochschule Esslingen
   University of Applied Sciences
   Flandernstr. 101
   73732 Esslingen am Neckar
   Germany

   Email: michael.scharf@hs-esslingen.de