LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: April 17, 2018

C. Gomez
UPC/i2CAT
J. Crowcroft
University of Cambridge
M. Scharf
Nokia

October 14, 2017

# TCP Usage Guidance in the Internet of Things (IoT) draft-ietf-lwig-tcp-constrained-node-networks-01

## Abstract

This document provides guidance on how to implement and use the Transmission Control Protocol (TCP) in Constrained-Node Networks (CNNs), which are a characterstic 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 tradeoffs. The objective is to help embedded developers with decisions on which TCP features to use.

## Status of This Memo

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

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

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

This Internet-Draft will expire on April 17, 2018.

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. [I-D.ietf-lwig-energy-efficient]).

At the application layer, the Constrained Application Protocol (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 the main reason why a CoAP over TCP specification is being developed [I-D.ietf-core-coap-tcp-tls].

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 [RFC7540] [RFC2616], 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 Queue Telemetry Transport (MQTT) and its lightweight variants.

TCP is a sophisticated transport protocol that includes many optional functionality and TCP options that improve performance. Many optional TCP extensions require complex logic inside the TCP stack and increase the codesize and the RAM requirements. However, many TCP extensions are not required for interoperability with other standard-compliant TCP endpoints. Given the limited resources on constrained devices, careful "tuning" of the TCP implementation can make an implementation more lightweight.

This document provides guidance on how to implement and 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 [RFC7414]. Similar guidance regarding the use of TCP in special environments has been published before, e.g., for cellular wireless networks [RFC3481].

## 2. Conventions used in this document

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

## 3. Characteristics of CNNs relevant for TCP

### 3.1. 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 on the 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 [I-D.ietf-lwig-energy-efficient], as well as minimization of the number of messages transmitted/received (and their size).

[RFC7228] lists typical network constraints in CNN, 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. CNN 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 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].

### 3.2. 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. CNNs may also follow the multihop topology [RFC6606]. One key use case for the use of TCP 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 constrained environments, there can be different types of devices [RFC7228]. For example, there can be devices with single combined send/receive buffer, devices with a separate send and receive buffer, or devices with a pool of multiple send/receive buffers. In the latter case, it is possible that buffers also be shared for other protocols.

When a CNN comprising one or more constrained devices and an unconstrained device communicate over the Internet using TCP, the communication possibly has to traverse a middlebox (e.g. a firewall, NAT, etc.). Figure 1 illustrates such 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, to have high amount of memory and processing power, and to 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.

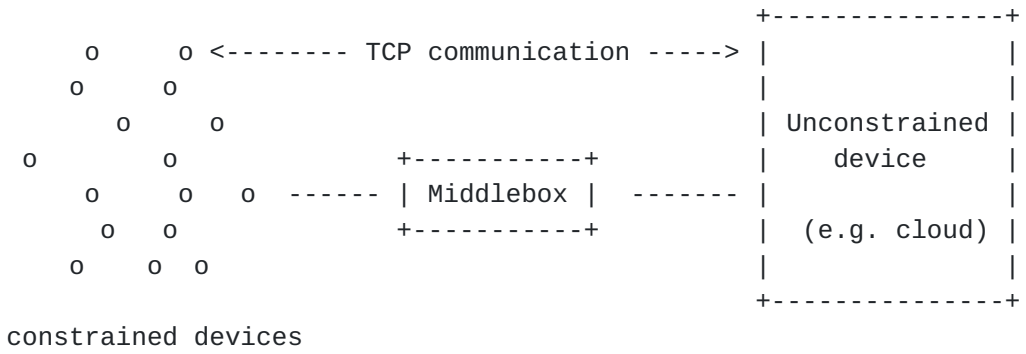


Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

## 3.3. 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 send (repeatedly) updates to the other endpoint. Not in every case there is 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.

## 4. TCP over CNNs

### 4.1. TCP connection initiation

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

### 4.2. Maximum Segment Size (MSS)

Some link layer technologies in the CNN space 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 [RFC2460], while IEEE 802.15.4 lacked fragmentation mechanisms. 6LoWPAN defines an IEEE 802.15.4 link MTU of 1280 bytes [RFC4944]. Other technologies, such as Bluetooth LE [RFC7668], ITU-T G.9959 [RFC7428] or DECT-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 / Token Passing (TP) [RFC8163], Narrowband IoT (NB-IoT) [I-D.ietf-lpwan-overview] or IEEE 802.11ah [I-D.delcarpio-6lo-wlanah], that do not suffer the same degree of frame size limitations as the technologies mentioned above. The MTU for MS/TP is recommended to 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.

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 MTU to 1280 bytes in order to avoid the need to support Path MTU Discovery [RFC1981].

An IPv6 datagram size exceeding 1280 bytes can be avoided by setting the TCP MSS not larger than 1220 bytes. (Note: IP version 6 is assumed.)

### 4.3. Window Size

A TCP stack can reduce the RAM requirements by advertising a TCP window size of one MSS, and also transmit at most one MSS of unacknowledged data. In that case, both congestion and flow control implementation is 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 one MSS can significantly affect performance. A stop-and-wait operation results in low throughput for transfers that exceed the lengths of one MSS, e.g., a firmware download. In addition, there can be interactions with the delayed acknowledgements (see Section 4.8).

Devices that have enough memory to allow larger TCP window size can leverage a more efficient error recovery using Fast Retransmit and Fast Recovery [RFC5681]. These algorithms work efficiently for window sizes of at least 5 MSS: If in a given TCP transmission of segments 1,2,3,4,5, and 6 the segment 2 gets lost, the sender should get an acknowledgement for segment 1 when 3 arrives and duplicate acknowledgements when 4, 5, and 6 arrive. It will retransmit segment 2 when the third duplicate ack arrives. In order to have segment 2, 3, 4, 5, and 6 sent, the window has to be at least five. With an MSS of 1220 byte, a buffer of the size of 5 MSS would require 6100 byte.

For bulk data transfers further TCP improvements may also be useful, such as limited transmit [RFC3402].

If CoAP is used over TCP with the default setting for NSTART in [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. For this use of CoAP, a maximum TCP window of one MSS will be sufficient.

### 4.4. RTO estimation

The Retransmission Timeout (RTO) estimation is one of the fundamental TCP algorithms. 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 lead 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.

[RFC6298] describes the standard TCP RTO algorithm. If a TCP sender uses very small window size and cannot use Fast Retransmit/Fast Recovery or SACK, the Retransmission Timeout (RTO) algorithm has a larger impact on performance than for a more powerful TCP stack. In that case, RTO algorithm tuning may be considered, although careful assessment of possible drawbacks is recommended.

As an example, an adaptive RTO algorithm for CoAP over UDP has been defined [I-D.ietf-core-cocoa] that has been found to perform well in CNN scenarios [Commag].

### 4.5. TCP connection lifetime

[[Note: future revisions will better separate what a TCP stack should support, or not, and how the TCP stack should be used by applications, e.g., whether to close connections or not.]]

#### 4.5.1. Long TCP connection lifetime

In CNNs, in order to minimize message overhead, a TCP connection should be kept open as long as the two TCP endpoints have more data to exchange or it is envisaged that further segment exchanges will take place within an interval of two hours since the last segment has been sent. A greater interval may be used in scenarios where applications exchange data infrequently.

TCP keep-alive messages [RFC1122] may be supported by a server, to check whether a TCP connection is active, in order to release state of inactive connections. This may be useful for servers running on memory-constrained devices.

Since the keep-alive timer may not be set to a value lower than two hours [RFC1122], TCP keep-alive messages are not useful to guarantee that filter state records in middleboxes such as firewalls will not be deleted after an inactivity interval typically in the order of a few minutes [RFC6092]. In scenarios where such middleboxes are present, alternative measures to avoid early deletion of filter state

records (which might lead to frequent establishment of new TCP connections between the two involved endpoints) include increasing the initial value for the filter state inactivity timers (if possible), and using application layer heartbeat messages.

### 4.5.2. Short TCP connection lifetime

A different approach to addressing the problem of traversing middleboxes that perform early filter state record deletion relies on using TCP Fast Open (TFO) [RFC7413]. In this case, instead of trying to maintain a TCP connection for long time, possibly short-lived connections can be opened between two endpoints while incurring low overhead. In fact, TFO allows data to be carried in SYN (and SYN-ACK) packets, and to be consumed immediately by the receceiving endpoint, thus reducing overhead compared with the traditional three-way handshake required to establish a TCP connection.

For security reasons, TFO requires the TCP endpoint that will open the TCP connection (which in CNNs will typically be the constrained device) to request a 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. Nevertheless, TFO is more efficient than frequently opening new TCP connections (by using the traditional three-way handshake) for transmitting new data, as long as the cookie update rate is well below the data new connection rate.

## 4.6. Explicit congestion notification

Explicit Congestion Notification (ECN) [RFC3168] may be used in CNNs. ECN allows a router to signal in the IP header of a packet that congestion is arising, for example when queue size reaches a certain threshold. If such a packet encapsulates a TCP data packet, an ECN-enabled TCP receiver will echo back the congestion signal to the TCP sender by setting a flag in its next TCP ACK. The sender triggers congestion control measures as if a packet loss had happened. In that case, when the congestion window of a TCP sender has a size of one segment, the TCP sender resets the retransmit timer, and will only be able to send a new packet when the retransmit timer expires [RFC3168]. Effectively, the TCP sender reduces at that moment its sending rate from 1 segment per Round Trip Time (RTT) to 1 segment per default RTO.

ECN can reduce packet losses, since congestion control measures can be applied earlier than after the reception of three duplicate ACKs (if the TCP sender window is large enough) or upon TCP sender RTO expiration [RFC2884]. Therefore, the number of retries decreases,

which is particularly beneficial in CNNs, where energy and bandwidth resources are typically limited. Furthermore, latency and jitter are also reduced.

ECN is particularly appropriate in CNNs, since in these environments transactional type interactions are a dominant traffic pattern. As transactional data size decreases, the probability of detecting congestion by the presence of three duplicate ACKs decreases. In contrast, ECN can still activate congestion control measures without requiring three duplicate ACKs.

### 4.7. TCP options

A TCP implementation needs to support options 0, 1 and 2 [RFC0793]. 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 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 [RFC1323], TCP Timestamps [RFC1323], Selective Acknowledgements (SACK) and SACK-Permitted [RFC2018]. Also other TCP options may not be required on a constrained device with a very lightweight implementation.

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. SACK is particularly useful for bulk data transfers. The receiver supporting SACK will need to manage the reception of possible out-of-order received segments, requiring sufficient buffer space. SACK adds 8*n+2 bytes to the TCP header, where n denotes the number of data blocks received, up to 4 blocks. For a low number of out-of- order segments, the header overhead penalty of SACK is compensated by avoiding unnecessary retransmissions.

Another potentially relevant TCP option in the context of CNNs is (TFO) [RFC7413]. As described in Section 4.5.2, TFO can be used to address the problem of traversing middleboxes that perform early filter state record deletion.

### 4.8. Delayed Acknowledgments

TCP Delayed Acknowledgements reduce the number of transferred bytes within a TCP connection, but they may increase the time until a sender may receive an ACK. For certain traffic patterns Delayed Acknowledgements may have a detrimental effect. Advanced TCP stacks may use heuristics to determine the maximum delay for an ACK. For CNNs, the recommendation depends on the expected communication patterns.

A device that advertises a single-MSS receive window should avoid use of delayed ACKs in order to avoid contributing unnecessary delay (of up to 500 ms) to the RTT [RFC5681], which limits the throughput and can increase the data delivery time.

A device that can send at most one MSS of data is significantly affected if the receiver uses delayed ACKs, e.g., if a TCP server or receiver is outside the CNN. One known workaround is to split the data to be sent into two segments of smaller size. A standard compliant TCP receiver will then immediately acknowledge the second segment, which can improve throughput. This "split hack" works if the TCP receiver uses Delayed Acks, but the downside is the overhead of sending two IP packets instead of one.

Also for larger windows, it may make sense to use a small timeout or disable delayed ACKs when traffic over a CNN is expected to mostly be small messages with a size typically below one MSS. For request-response traffic between a constrained device and a peer (e.g. backend infrastructure) that uses delayed ACKs, the maximum ACK rate of the peer will be typically of one ACK every 200 ms (or even lower). If in such conditions the peer device is administered by the same entity managing the constrained device, it is recommended to disable delayed ACKs at the peer side.

In contrast, delayed ACKs allow to reduce the number of ACKs in bulk transfer type of traffic, e.g. for firmware/software updates or for transferring larger data units containing a batch of sensor readings.

### 4.9. 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.

## 5. Security Considerations

Best current practise for securing TCP and TCP-based communication also applies to CNN. As example, use of Transport Layer Security (TLS) is strongly recommended if it is applicable.

There are also TCP options which can improve TCP security. Examples include the TCP MD5 signature option [RFC2385] and the TCP Authentication Option (TCP-AO) [RFC5925]. However, both options add overhead and complexity. The TCP MD5 signature option adds 18 bytes to every segment of a connection. TCP-AO typically has a size of 16-20 bytes.

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

## 6. Acknowledgments

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

The authors appreciate the feedback received for this document. The following folks provided comments that helped improve the document: Carsten Bormann, Zhen Cao, Wei Genyu, Ari Keranen, Abhijan Bhattacharyya, Andres Arcia-Moret, Yoshifumi Nishida, Joe Touch, Fred Baker, Nik Sultana, Kerry Lynn, Erik Nordmark, Markku Kojo, and Hannes Tschofenig. Simon Brummer provided details on the RIOT TCP implementation. Xavi Vilajosana provided details on the OpenWSN TCP implementation. Rahul Jadhav provided details on the uIP TCP implementation.

## 7. Annex. TCP implementations for constrained devices

This section overviews the main features of TCP implementations for constrained devices.

### 7.1. uIP

uIP is a TCP/IP stack, targetted for 8 and 16-bit microcontrollers. uIP has been deployed with Contiki and the Arduino Ethernet shield. A code size of ~5 kB (which comprises checksumming, IP, ICMP and TCP) has been reported for uIP [Dunk].

uIP uses same buffer both incoming and outgoing traffic, with 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.

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

Contiki uses the "split hack" technique (see Section 4.8) to avoid delayed ACKs for senders using a single MSS.

### 7.2. lwIP

lwIP is a TCP/IP stack, targetted 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, IP, ICMP and TCP), and a TCP code size of ~9 kB to ~14 kB [Dunk].

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 buffering of incoming and outcoming data. Other implemented mechanisms comprise slow start, congestion avoidance, fast retransmit and fast recovery. SACK and Window Scale have been recently added to lwIP.

### 7.3. RIOT

The RIOT TCP implementation (called GNRC TCP) has been designed for Class 1 devices [RFC 7228]. The main target platforms are 8- and 16-bit microcontrollers. 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. GNRC TCP uses a single-MSS window size, which simplifies the 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.

### 7.4. OpenWSN

The TCP implementation in OpenWSN is mostly equivalent to the uIP TCP implementation. OpenWSN TCP implementation only supports the minimum state machine functionality required. For example, it does not perform retransmissions.

### 7.5. TinyOS

TODO: To be verified

TinyOS has an experimental TCP stack that uses a simple nonblocking library-based implementation of TCP. 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 and otherwise drop the segment. A send buffer is provided so that the TCP implementation can automatically retransmit missing segments.

### 7.6. Summary

| | | uIP | lwIP orig | lwIP 2.0 | RIOT | OpenWSN | TinyOS |
|---|---|---|---|---|---|---|---|
| Memory | Data size | * | * | * | * | * | * |
| | Code size (kB) | < 5 | ~9 to ~14 | * | * | * | * |
| | Window size(MSS) | 1 | Multiple | Multiple | 1 | 1 | Multiple |
| TCP C | Slow start | No | Yes | Yes | No | No | Yes |
| | Fast rec/retx | No | Yes | Yes | No | No | Yes |
| | Keep-alive | No | * | * | No | No | No |
| TCP options | TFO | No | No | * | No | No | No |
| | ECN | No | No | * | No | No | No |
| | Window Scale | No | No | Yes | No | No | No |
| | TCP timestamps | No | No | Yes | No | No | No |
| | SACK | No | No | Yes | No | No | No |

```
   |        +----------------+-------+---------+---------+------+---------
+--------+
   |        | Delayed ACKs   |  No   |   Yes   |   Yes   |  No  |   No    |
No   |
   +--------+----------------+-------+---------+---------+------+---------
+--------+
```

Figure 2: Summary of TCP features for differrent lightweight TCP implementations.

TODO: Add information about RAM requirements (in addition to codesize)

## 8. **Annex.** Changes compared to previous versions

RFC Editor: To be removed prior to publication

### 8.1. **Changes compared to -00**

- Changed title and abstract
- Clarification that communcation with standard-compliant TCP endpoints is required, based on feedback from Joe Touch
- Additional discussion on communication patters

- Numerous changes to address a comprehensive review from Hannes Tschofenig
- Reworded security considerations
- Additional references and better distinction between normative and informative entries
- Feedback from Rahul Jadhav on the uIP TCP implementation
- Basic data for the TinyOS TCP implementation added, based on source code analysis

## 9. References

### 9.1. 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>.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, DOI 10.17487/RFC1323, May 1992, <https://www.rfc-editor.org/info/rfc1323>.

[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>.

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[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>.

[RFC3402] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm", RFC 3402, DOI 10.17487/RFC3402, October 2002, <https://www.rfc-editor.org/info/rfc3402>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

## 9.2. Informative References

[Commag] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP Congestion Control for the Internet of Things", IEEE Communications Magazine, June 2016.

[Dunk] A. Dunkels, "Full TCP/IP for 8-Bit Architectures", 2003.

[ETEN] R. Krishnan et al, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks", Computer Networks 2004.

[I-D.delcarpio-6lo-wlanah] Vega, L., Robles, I., and R. Morabito, "IPv6 over 802.11ah", draft-delcarpio-6lo-wlanah-01 (work in progress), October 2015.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-09 (work in progress), May 2017.

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-01 (work in progress), March 2017.

[I-D.ietf-lpwan-overview]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-07 (work in progress), October 2017.

[I-D.ietf-lwig-energy-efficient]
Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, "Energy-Efficient Features of Internet of Things Protocols", draft-ietf-lwig-energy-efficient-07 (work in progress), March 2017.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <https://www.rfc-editor.org/info/rfc1981>.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <https://www.rfc-editor.org/info/rfc2385>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <https://www.rfc-editor.org/info/rfc2616>.

[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>.

[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>.

[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>.

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <https://www.rfc-editor.org/info/rfc6092>.

[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>.

[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>.

[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>.

[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>.

## Authors' Addresses

Carles Gomez
UPC/i2CAT
C/Esteve Terradas, 7
Castelldefels 08860
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
Nokia
Lorenzstrasse 10
Stuttgart, 70435
Germany

Email: michael.scharf@nokia.com