MIPRO 2018, May 21-25, 2018, Opatija Croatia

On HTTP Performance in IoT Applications: An Analysis of Latency and Throughput

Wolfgang Bziuk, Cao Vien Phung, Jasenka Dizdarević and Admela Jukan
Technische Universität Braunschweig, Germany
Email:{w.bziuk, c.phung, j.dizdarevic, a.jukan}@tu-bs.de

Abstract—**In recent years, Internet of Things (IoT) industry has developed rapidly thanks to the rise of the smart objects and devices with processing, sensing and actuating capabilities. In order to satisfy a broad range of applications, a variety of application layer protocols has been taken into consideration, since IoT still does not have widely accepted standard protocol. One of prime candidate solutions is HTTP, as a well-known, fundamental client-server protocol and the protocol that is the most compatible with existing network infrastructure. Due to strict IoT specific requirements, however, a detailed characterization of the performance of HTTP for IoT applications is required. This paper presents a detailed analysis of throughput and latency for HTTP/1.1 with pipelining, by evaluating the impact that pipelining factor has on the latency. Based on these performance results, we analyze whether this HTTP/1.1 model should be implemented in IoT systems with significant latency constraints.**

I. Introduction

With the growth of Internet of Things (IoT) industry and increasing number of smart objects and devices in recent years, one of the main factors that will determine the IoT performance will be the communication between these devices. To this end, the selection of the application layer protocol (messaging protocol) that is able to satisfy a broad range of IoT applications is critical. There are many potential IoT protocol candidates, and attempts are being made towards their standardization, such as Message Queue Telemetry Transport (MQTT) [1], Constrained Application Protocol (CoAP) [2], and Representational State Transfer (REST) HTTP [3]. In an attempt to capture new trends in IoT, major industry is also looking for solutions. For instance, Google developed an experimental protocol SPDY [4] which was used as a base for a new version of HTTP protocol HTTP/2.0 [5]. The basic research in this area is however still under development and as of today there is no enough evidence that there is a winning solution or a replacement of the current HTTP/1.1 model.

Despite the fact that HTTP is designed without consideration of the resource constrained devices used in IoT it remains widely accepted within the existing network infrastructure and it is unlikely that there will be an IoT system solution that will not rely on HTTP in some degrees. This is why it is very important to tackle some of the challenges of using HTTP in IoT, such as:

- Latency: A big impediment in adopting HTTP lies in the long time needed for the server to receive and process a request from the client. This latency is a consequence of both HTTP and its underlying TCP.
- Energy/power consumption: This is particularly important issue for resource constrained devices, where large uncompressed and redundant HTTP headers cause high network usage and energy consumption.
- No server push option: As a request/response protocol, a HTTP server has to wait for an explicit request from the client before a data transfer is initialized.
- Head-of-line blocking (HOL) problem: HTTP/1.1 allows pipelining to request multiple objects over the same TCP connection. But since servers must respond to requests in order, an early request for a large object as a lost request can delay all subsequent pipelined requests.

It should be noted that even with all the issues, – HTTP/1.1 is the most stable and widely acceptable solution. From a practical point of view, it maybe even easier to adapt than to implement other protocols. For these reasons, this paper sets the goal to model the HTTP throughput and latency for IoT application, in order to assess the performance. Our focus is on the performance of HTTP/1.1 with pipelining, and the effect of the pipelining factor has on the latency.

As we will show with results HTTP/1.1 with pipelining does perform better than without pipeling, which is the regularly accepted model. But, the improvement is not worth the difficulties that come with its implementation and there are other better IoT solutions.

II. Related Work

Several models of HTTP have been proposed already. A first analytical model was presented in [6] and used to examine the interaction of HTTP with some underlying transport protocols and in a similar way, [7] conducted an analysis of HTTP performances over satellite channels. In [8], the performance of HTTP/1.1 with and without pipelining is evaluated by an approximate mathematical model using several TCP-related parameters and a realistic model for a web-page.

Analytical models of HTTP rely on TCP models, that have an extensive literature. For an infinite data source the stationary throughput in the congestion avoidance phase of TCP has been analysed by some authors, and well established results for RENO have been derived in [9]. This work neglects the effect of the initial TCP slow start. This gap has been closed in [10]. Based on the work of [9], the authors derived an analytical model for TCP latency which considers TCP's connection establishment, Slow Start (SS) and Congestion Avoidance (CA).

However with a current modeling, standard HTTP behaviours can not easily capture new trends in IoT. This is a reason some authors have decided to address limitations of HTTP and analyse its alternatives such as SPDY and HTTP/2.0. In [11] authors have compared web latency of these two protocols with existing HTTP/1.1. The experimental results showed that implementation of these protocols at the server side may improve the performance and reduce the web latency, however, but no significant improvement has been noticed on the client side. Also in [5] authors tested performances of HTTP/1.1 and HTTP/2.0 and noticed faster page load time with HTTP/2.0. All of this works assumed HTTP without pipelining.

III. Analytical Model

A. Assumptions and Prerequisite

1) HTTP Overview: The typical web-page hosted by the server consists of a base-page (index-file) for the top-level HTML with a number of M embedded objects. In practice [12], first the client will only request the base-page with expected size of $E\{F_0\}$ MSS from the server. Since HTTP uses TCP, the maximum segment size (MSS) or synonymous segment is used to specify the file size. Next requests are sent for the embedded objects, where each object contains an average number of $E\{F_s\}$ MSS. Let us assume an expected number $E\{M\}$ of embedded objects. Then, with $D_{S_M} = E\{D_S(M)\} = E\{M\} \cdot E\{F_s\}$, the page size is given by

$$E\{PS\} = E\{F_0\} + E\{M\} \cdot E\{F_s\} = E\{F_0\} + D_{S_M} \quad (1)$$

Let us assume HTTP/1.1 uses a single persistent TCP connection, where embedded objects are pipelined according to a Pipeline Factor $N, N \leq N_{max}$. The maximum number of pipelined requests sent simultaneously is typically limited by the server to $N_{max} = 4, ..., 8$. Examples for the HTTP/TCP cross-layer actions at the client/server side are shown in Fig.1 and Fig.2 for a web-page with a base-page of size $F_0 = 3$ segments, which contains $M = 4$ embedded objects, each of size $F_s = 13$ segments. Fig.1 represents the pipelined transfer, where after the initial request (blue segment) the base-page (3 green segments) is downloaded. Let RTT be the average round trip time between the client and server, which is very large compared to the transmission time of a segment. In the detailed TCP-subgraph of Fig.1 this allows us to horizontally draw the number of segments sent by TCP per round (per RTT). Next, all $M = 4$ objects are requested at once, as illustrated by the blue segments in the figure. The four objects are downloaded as if one file of size $M \cdot F_s$ is continuously transmitted, as shown by the green segments. Fig.2 illustrates the HTTP with pipelining using at most $N = 2$ requests at a time. When $N = 1$, we have the special case of HTTP/1.1 without pipelining, where only one request is sent at the same time and the response has to be successfully downloaded before the next request can be sent. Depending on the pipelining factor we observe a different behaviour at the TCP layer, thus the TCP protocol has to be considered in more detail.

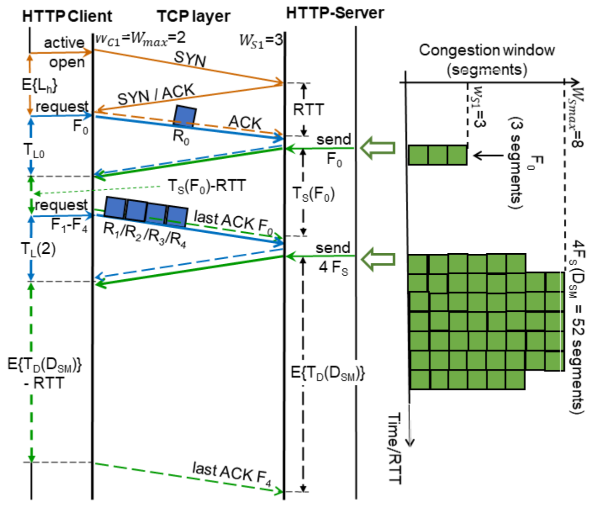


Fig. 1: HTTP/1.1 with 4 pipelined requests

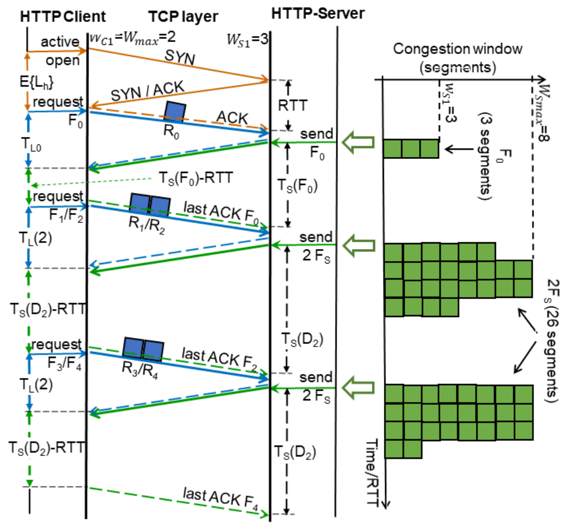


Fig. 2: HTTP/1.1 with 2 pipelined requests

2) TCP Overview: To determine the latencies seen at the HTTP client and server side we use results derived in [10] which allow TCP behaviour analysis and can be easily extended to different TCP versions. Let us give an overview and reference of the expressions in form (Equation, [10]). For the TCP connection establishment process, a loss rate p_s for the small SYN/ACK packets in either direction and a SYN timeout, T_s, of 3 seconds is assumed. To show the dependency on the TCP parameters, the expected latency is referred to as

$E\{L_h(RTT, T_s, p_s)\}$, (4, [10]), or simple

$$E\{L_h\} = RTT + 2 \cdot T_s \cdot (\frac{1-p_s}{1-2p_s} - 1) \tag{2}$$

For the TCP data transfer of a fixed number of d data segments from one host (client or server) to another the data transfer latency follows to be

$$E\{T_D(d)\} = E\{T_{SS}(d)\} + E\{T_{Loss}(d)\} + E\{T_{CA}(d)\}. \tag{3}$$

This takes into account the contributions due to the slow start, $E\{T_{SS}\}$, loss recovery latency at the end of the slow start, $E\{T_{Loss}\}$, and the time for transmitting data in congestion avoidance, $E\{T_{CA}\}$. Each latency term is defined as the time interval that starts when an application places data in the send buffer and ends when the TCP receives an acknowledgement (Ack) for the last byte in the buffer. Each segment is of common size (MSS and Header) and p is the data segment loss rate. Because the data transfer and connection establishment are taken place on the same TCP chanel, so we assume $p = p_s$. For loss probability values we consider the interval $[10^{-2}, 10^{-1}]$ commonly used for low-SNR wireless links that are being used in IoT applications [13]. Since IoT applications require a low latency, the $delayedAck$ option of TCP is disabled. The main behaviour of TCP is controlled by the congestion window, $cwnd$. This is limited by W_{max}, where W_{max} may depend on the advertised window. For d segments TCP starts in Slow Start with initial $cwnd$ of w_1 segments. The expected number of data segments sent before a loss terminates the slow start can be determined by $E\{D_{SS}(d)\} = E\{D_{SS}(d,p)\}$, (5, [10]), where the probability of this loss is given by $l_{SS}(d) = 1-(1-p)^d$. At the end of slow start the expected value of the window size is estimated as $E\{W_{SS}(d)\} = min\{(E\{D_{SS}(d)\}+w_1)/2, W_{max}\}$, (11, [10]). The time to send $E\{D_{SS}(d)\}$ data segments in slow start before a loss occurs is approximately given by $E\{T_{SS}(d)\} = E\{T_{SS}(d, p, RTT, w_1, W_{max})\}$, (15, [10]), and followed by the recovery time due to segment retransmission done by fast retransmission or retransmission timeouts (RTOs), $E\{T_{Loss}(d)\} = E\{T_{Loss}(d, p, RTT, w_1, W_{max}, T_O)\}$, (20, [10]). T_O is the average duration of the first TCP timeout in a sequence of these. The data left after slow start is approximately $E\{D_{CA}(d)\} = max\{0, d - E\{D_{SS}(d)\}$ and sent using CA. Let $R = R(p, RTT, W_{max}, T_O)$, (22, [10]), be the throughput for an unlimited TCP data source first derived in [9], which is used to approximate the expected time to send the remaining data as $E\{T_{CA}(d)\} = E\{D_{CA}(d)\}/R$.

B. Model of HTTP/1.1 with pipelining

In this section we derive an expression for the download time (latency) of a web-page seen at the client side and the achievable throughput. Vaderna et. al. [8] have derived an approximation only for the pipeline factors $N = 1$ and $N = M$. They do not consider the download of the HTML base-page and only take into account the latency observed at the server side. Based on this work, we will derive a more general analysis for $1 \leq N \leq N_{max}$ and most notable, we also take into account the latency at the client side.

The behaviour of the TCP layer depends on the progress of the congestion window $cwnd$, but due to the properties of the HTTP/1.1 protocol this behaviour is quite different for the client (C) and server (S) side. Let us denote the TCP initial window used by slow start at the HTTP client and server side as w_{C1} and w_{S1}, and in the same way an average congestion window as W_C and W_S.

1) Modelling of the client side: The analysis derived in [10] assumes that the application can immediately fill the $cwdn$ offered by TCP with segments, which is not the case for the client side. As shown in Fig.2, the client sends at most a fixed number of N requests to start a download cycle at the server, where we assume conservatively that each request is transmitted using a separate segment with MSS_C bytes. TCP typically starts with an initial window with the range $w_1 = 2, ..., 4$ (see RFC5681, implementations already use initial windows up to 10). Due to the initial slow start phase of TCP, after N sent and acknowledged requests the $cwnd$ has already increased to $W_C = w_{C1} + N > N$ (no delayed ACK). Thus, already after the first download cycle the client can not fill the available $cwdn$, and requires the adapted results derived in [10].

For the first request of the HTML base-page a single segment is sent. With probability p this segment is lost and an RTO will occur. The latency resulting from the retransmission including possible further RTOs has been derived in [9] and is given by $E\{Z^{TO}\}$, (18, [10]), thus the expected latency seen at the client side follows to be

$$T_{L0} = RTT + p \cdot E\{Z^{TO}\} \tag{4}$$

Next the latency seen at the client side for a cycle of N consecutively sent requests is derived. Since our focus is on IoT, we want to reduce the latency which is supported by choosing an initial $cwnd$ equal to the pipeline factor, e.g. $w_{C1} = N$. Furthermore, by setting $w_{C1} = W_{max} = N$, the analysis presented in [10] can be applied. This includes the loss of segments and most importantly, due to the small number of segments sent, the probability for a RTO is high, which results in high latencies. Furthermore, our models assume that the download cycle at the server side starts after the transmission of all N requests are successful, thus the important problem of the head-of-line blocking of requests is taken into account.

Due to the retransmission feature of TCP, at the end of each request cycle, the client will see a $cwdn$ whose utilization is limited to $W_{max} = N$ due to the pipeline factor. This enables us to consider each cycle of N client requests independent from other cycles. In summary, according to Eq.(3) using $d = N$, the latency for each request cycle can be calculated as

$$T_L(N) = E\{T_D(N)\}, N = 1, ..., N_{max} \tag{5}$$

It should be noted, that for $N = 1$ we get $T_L(1) = T_{L0}$.

2) Modelling of the server side - HTML base-page: Let us analyse the download time seen at the server side for the base-page independently from the download of embedded objects. Eq.(3) indicates the latency for the transmission of d segments. Similar to the approach used in [8], we assume that Eq.(3) can also be used to derive the delay for an average number of $d_0 = E\{F_0\}$ segments. With Eq.(3) and $E\{T_D(d_0)\} = T_S(F_0)$ we get

$$T_S(F_0) = E\{T_{SS}(d_0)\} + E\{T_{Loss}(d_0)\} + E\{T_{CA}(d_0)\} \quad (6)$$

Following [10], $E\{D_{SS}(d_0)\}$ segments can be transmitted before a loss terminates slow start and after loss recovery the remaining data $E\{D_{CA}(d_0)\} = \lfloor d_0 - E\{D_{SS}(d_0)\}\rfloor$ is sent using congestion avoidance. Especially for a small number of segments d_0, the congestion avoidance phase is omitted and we have $E\{D_{CA}(d_0)\} = 0$, i.e. no segment is left after slow start and TCP will remain in slow start with final average window given by $E\{W_{SS}(d_0)\}$. Furthermore, $E\{T_{CA}(d_0)\} = 0$ in Eq.(6). Otherwise, if $E\{D_{CA}(d_0)\} > 0$, we use $W_{CA}(p) = 1 + \sqrt{1 + \frac{8(1-p)}{3p}}$, which is the expected congestion window at time of loss event in congestion avoidance, see (13, [9]).

3) Modelling of the server side - embedded objects: The download of the embedded objects is analysed in the same way as done for the base-page taking into account the existing state of TCP. If $E\{D_{CA}(d_0)\} = 0$, we remain in slow start with an initial window set to $w_{S1} = E\{W_{SS}(d_0)\}$. On the other side, if $E\{D_{CA}(d_0)\} > 0$, we assume that TCP stays in congestion avoidance with $W_{CA}(p)$ given above. In the example shown in Fig.1, TCP starts in the first round in slow start with w_{S1} and later continues in congestion avoidance limited by the maximum window size W_{Smax}. Thus, the analytical model derived in [10] can be applied without changes.

Let us first consider the case, where all requests to embedded objects sent by the client are pipelined at once, i.e. $N = M$, which is shown in Fig.1. Since all objects can be considered as a single file of average size $D_{S_M} = M \cdot E\{F_s\}$, its segments can be sent continuously and the $cwdn$ at the server side determines the number of segments sent per RTT round. Using Eq.(1)-(3), the download time for D_{S_M} segments is given by

$$\begin{aligned} E\{T_D(D_{S_M})\} =& E\{T_{SS}(D_{S_M})\} + E\{T_{Loss}(D_{S_M})\} \\ &+ E\{T_{CA}(D_{S_M})\} \end{aligned} \quad (7)$$

It should be noted, that for $E\{D_{CA}(d_0)\} > 0$ the first two terms in Eq.(7) related to slow start are omitted. Furthermore, the related throughput can be approximated as

$$\gamma_S(D_{S_M}) = \frac{D_{S_M}}{E\{T_D(D_{S_M})\}} = \frac{W_S \cdot E\{R_S\}}{E\{T_D(D_{S_M})\}} = \frac{W_S}{RTT} \quad (8)$$

which also allows to derive the average window size, W_S, and the average number of used RTT rounds, $E\{R_S\}$.

For a pipeline factor $N < M$ the server will only continuously send blocks of N objects with size $E\{D_S(N)\} = N \cdot E\{F_s\}$ and wait for the next download until the next requests arrive, which is demonstrated in Fig.2 for $N = 2$. Thus, there are $I_S = \lceil M/N \rceil$ download cycles. Only to simplify the notation, let us restrict to use cases with $M = I_S \cdot N$. At the end of a download cycle, the downloaded file does not use the whole available $cwdn$. As a consequence, the analysis derived in [10] will underestimate the number of RTT rounds required for the download. This is demonstrated in Fig.2, where at the end of the first download cycle the available window is not used by 4 segments. This increases the total download time by one round (RTT), as can be seen by a comparison with Fig.1.

For $N = 1$ this effect has been solved approximately in [8], which we will generalize to arbitrary N. Although the different download cycles are separated by the latency due to request arrivals and some segments do not fill the window in the last round, this has only a small impact on the evolution of the average $cwdn$, since the transmission of the next data block continues with the final $cwdn$ of the preceding one. This allows to use, as approximation, the average $cwdn$ W_S derived for $E\{D_{S_M}\}$ segments transmitted as continuous stream.

Consequently, following [8], x segments are transferred in $\lceil x/W_S \rceil$ RTT rounds. Furthermore, let us assume that the number of segments of N objects with average size $E\{D_S(N)\}$ is distributed according to the density $f_{D_N}(x)$. Then, the average number of RTT rounds required for the download of N objects can be calculated as

$$\widehat{R}(D_N) = \int_0^{\infty} \left\lceil \frac{x}{W_S} \right\rceil f_{D_N}(x)dx. \quad (9)$$

The average latency seen at the server follows to be

$$T_S(D_N) = \widehat{R}(D_N) \cdot RTT, N = 1, ..., M-1. \quad (10)$$

Let us assume, that a single object is exponentially distributed with mean $F_S = E\{F_s\}$. Consequently, a block of N objects with average size $E\{D_S(N)\} = N \cdot F_S$ follows an Erlang-N distribution given as $f_{D_N}(x) = \frac{x^{N-1} e^{-x/F_s}}{(F_s)^N (N-1)!}$. This yields a solution of Eq.9 given as

$$\widehat{R}(D_N) = \sum_{i=0}^{\infty} e^{-\frac{iW_S}{F_S}} \sum_{k=0}^{N-1} \frac{(\frac{iW_S}{F_S})^k}{k!} \quad (11)$$

For the special case $N = 1$ we get in agreement with [8], a closed form solution given as

$$\widehat{R}(D_1) = \widehat{R}(F_S) = [1 - e^{-\frac{W_S}{F_S}}]^{-1} \quad (12)$$

C. Latency and throughput of HTTP/1.1 with pipelining

As represented in Fig.2, for $M = I_S \cdot N$ and $N = 1, ..., N_{max}$, the overall latency seen at the client for a download of a complete web-page, T_{PS}, can be estimated as

$$T_{PS}(N) = E\{L_h\} + T_{CL}(N) + T_{SL}(N) - (I_S + 1) \cdot RTT \quad (13)$$

where parts observed at the client and server are given as

$$\begin{aligned} T_{CL}(N) &= T_{L0} + I_S \cdot T_L(N) \\ T_{SL}(N) &= T_S(F_0) + I_S \cdot T_S(D_N) \end{aligned} \quad (14)$$

Since Eq.(13) describes the overall latency at the client side, each delay term evaluated at the server side is reduced by the RTT. This is due to the fact, that the first segment (last ACK) for a block of downloaded data at the server side overlaps with the last ACK (first request) send by the client side, as demonstrated in Fig.2. Neglecting TCP connection setup the throughput γ_{PS} is often characterized by an average window given as

$$W_{PS} = \frac{E\{PS\}}{(T_{PS} - E\{L_h\})} \cdot RTT = \gamma_{PS} \cdot RTT \qquad (15)$$

IV. Numerical results

In this section we analyse the impact of the pipelining factor N on the latency of HTTP/1.1. For our examples we chose an average request size of $MSS_C = 800$ bytes [4]. Some parameters for TCP usage in IoT are taken from [14]. We use $MSS = 1460$ bytes for the file downloads and for the TCP timeout event $T_0 = 1s$. Furthermore, the server side's TCP initial is $w_{S1} = 5$. For the HTML base-page we assume an average value of $E\{F_0\} = 5MSS$ as this is usually the index file with links to the embedded objects. Ohter TCP parameters are selected from [10], such as maximum window size $W_{Smax} = 32$, and $RTT = 0.1s$. From [15] two examples are selected. The average response size $E\{F_S\}$ is $8kB$ for the SVG example, and $20kB$ for JS. For both we assume that the number of objects in one page is $M = 12$. For JS the total size of the embedded objects is roughly given by $D_{S_M} = 170MSS$ (see Eq.(1)), which is 2.4 times larger compared to SVG.

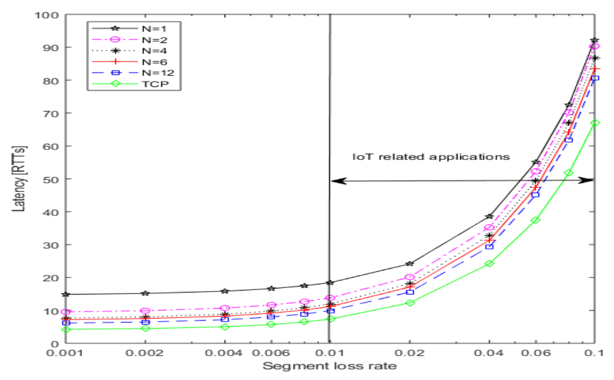
Figure 3 shows the latency to download the web-page seen at the client side normalized by the RTT. The latency is calculated by using Eq.(13), where the delay to download the embedded objects is given by (7) for $N = M$ and (10) for $N < M$. HTTP with $N = M = 12$ pipelined requests achieves a lower latency compared to the remaining scenarios. First we consider an example with a small loss rate of $p = 0.001$ for the SVG example shown in Fig. 3a. Compared to pipelining with $N = 12$, a pipeline factor of $N = 1$ increases the latency at min by 12 RTTs due to the additional requests. By neglecting the connection setup ($\sim$ RTT) the remaining latency does not increase proportionally to the 2.5 times larger web-page of JS, as shown in Fig. 3b. For the low loss rate and due to the small sized web-pages most of the segments are sent during the slow start and the $cwnd$ is restricted only by W_{Smax}. This increases the latency of JS by ~ 2 RTT which can be omitted using a larger W_{Smax}. In case of a high loss rate of $p = 0.1$, slow start already finishes after ~ 9 segments in both examples, thus the data transfer is controlled by congestion avoidance. The high loss rate is the cause of the large number of RTOs, which is the main reason for the drastic increase of the latency. For IoT solutions just looking at these results we can conclude that latency is lower with a higher pipeline factor. However, the issue of HOL blocking that was not taken into consideration presents a very big problem when implementing HTTP with pipelining. Comparing these latency results to other messaging protocols such as MQTT and CoAP [16] with which it is already possible to achieve very low latency without having to worry about HOL blocking, the improvement is not of a significant value.

Finally, it is important to compare the performance of HTTP with that of a simple TCP transmission, which is unaffected by any additional protocol overhead. The latency is again given by (14), where we set $I_S = 0$ and use the whole page-size $E\{PS\}$ instead of F_0. It should be noted, that the reduction by one RTT takes into account, that the TCP connection is initialized at the client side and the latency does not consider the last ACK sent back to the server, as represented in Fig.1. The negative effect of the overhead introduced by HTTP/1.1 strongly increases with decreasing pipelining factor, which shows a similar behaviour inside the range of considered loss rates. A further reduction seems only to be possible by introducing a server push service, which partially reduces the latency of the client initialized TCP connection. The usual applied methods of parallel TCP connections can also reduce the latency, but due to the limited resources of IoT devices, this seems too impractical.

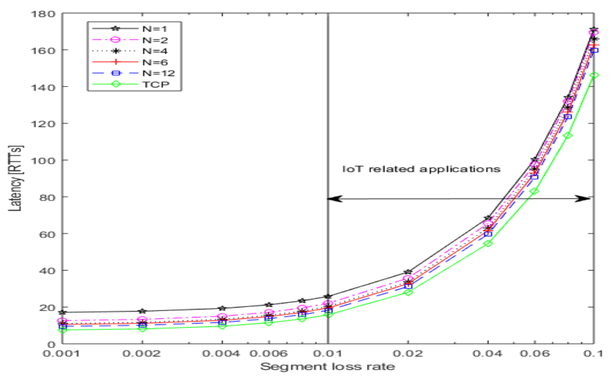
Figure 4 shows the average window size in MSS, which is computed by using (15) and can be viewed as a normalized value of the throughput observable at the client side. Since the TCP connection setup time (see (2)) only depends on the loss rate, it is neglected in (15) to allow a comparison regarding the size of the web-page. As discussed above, in the range of small loss rates ($p = 0.001$), slow start dictates the throughput behaviour and again, with decreasing pipelining factor N, the throughput drops down considerably due to the additional latency introduced by the client requests. For higher loss rates congestion avoidance is increasingly responsible for the data transfer. In this area TCP's average window is approximately given by $\bar{W}_{TCP}(p) \approx 0.75W_{CA}(p) \sim \sqrt{\frac{3}{2p}}$, [9], which explains the principle slope of the decreasing window observable for all pipelining factors. As example, for $p = 0.1$ we get independent of the size of the web-page $\bar{W}_{TCP}(p) \approx 3.9$, which is further reduced by the latency inserted by HTTP. In this loss area, as observable for both examples shown in Fig. 4, the performance of HTTP is dictated by the loss behaviour of TCP congestion avoidance and the effects of the pipelining factor vanishes. As expected, the exclusive use of TCP determines an upper bound for the achievable throughput.

V. Conclusion

HTTP 1.1 as a messaging protocol in IoT comes with many problems. The most widely accepted model without pipelining with its utilization of TCP means that established connections are released on every access, and often it requires multiple connection establishments for communication to be completed. In IoT this kind of communication causes overhead and consumption of network resources, so we decided to analyse HTTP with pipelining. By using HTTP 1.1 with pipelining we have shown that the latency can be reduced and throughput improved.

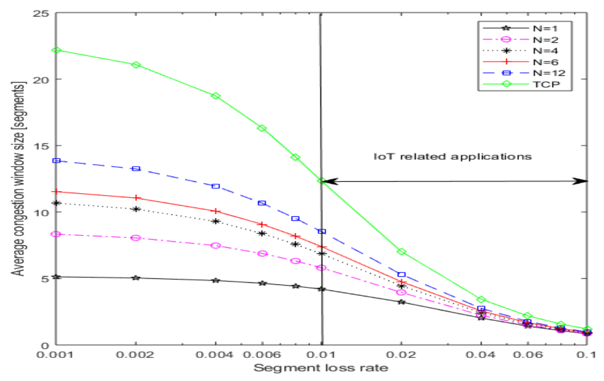


(a) average response size = $8kB$

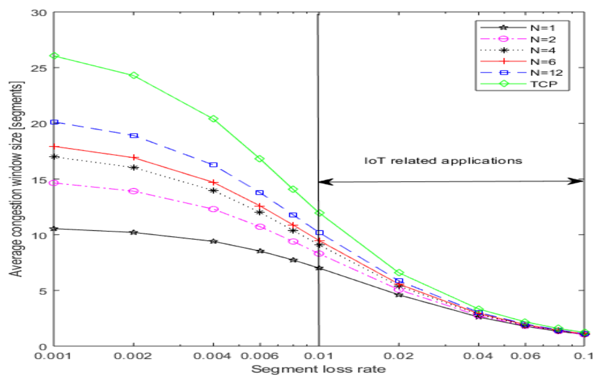


(b) average response size = $20kB$

Fig. 3: HTTP/1.1 latency depending on segment loss rate and pipelining factor.



(a) average response size = $8kB$



(b) average response size = $20kB$

Fig. 4: Average congestion window size depending on segment loss rate and pipelining factor.

But these improvement with pipelining comes with a set of other problems such as complex implementation and HOL blocking. With that in mind we think that the future research should be aimed at new messaging protocols and a new HTTP version, HTTP/2.0.

Acknowledgment

This work has been partially performed in the framework of mF2C project funded by the European Union's H2020 research and innovation programme under grant agreement 730929.

References

[1] OASIS, "MQTT Version 3.1.1," *OASIS Standard*, p. 81, 2014.

[2] Z. Shelby and C. Hartke, K. Bormann, "rfc7252, The Constrained Application Protocol (CoAP)," pp. 1–112, 2014.

[3] C. Severance, "Roy T. Fielding: Understanding the REST Style." *Computer*, vol. 48, no. 6, pp. 7–9, 2015.

[4] T. chromium projects, "Spdy: An experimental protocol for a faster web," 2012. [Online]. Available: http://dev.chromium.org/spdy/spdy-whitepaper

[5] H. De Saxce, I. Oprescu, and Y. Chen, "Is HTTP/2 really faster than HTTP/1.1?" *IEEE INFOCOM*, pp. 293–299, 2015.

[6] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the performance of http over several transport protocols," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 616–630, Oct. 1997.

[7] H. Kruse, M. Allman, J. Griner, and D. Tran, "Experimentation and modelling of http over satellite channels," *Inter. Jour. of Satellite Communications*, vol. 16, no. 1, pp. 51–68, 2001.

[8] P. Vaderna, E. Stromberg, and T. Elteto, "Modelling performance of http/1.1," in *Global Telecommunications Conference*, USA, 2003.

[9] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: A simple model and its empirical validation," *ACM SIGCOMM 98*, vol. 28, no. 4, pp. 303–314, 1998.

[10] N. Cardwell, S. Savage, and T. Anderson, "Modeling tcp latency," *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 2000.

[11] N. Naik and P. Jenkins, "Web protocols and challenges of Web latency in the Web of Things," *International Conference on Ubiquitous and Future Networks, ICUFN*, vol. 2016-August, pp. 845–850, 2016.

[12] B. Totty, D. Gourley, M. Sayer, A. Aggarwal, and S. Reddy, *Http: The Definitive Guide*. O'Reilly & Associates, Inc., 2002.

[13] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G. E. Corazza, "Internet of things application layer protocol analysis over error and delay prone links," in *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, Sept 2014, pp. 398–404.

[14] C. Gomez, J. Crowcroft, and M. Scharf, "Tcp usage guidance in the internet of things (iot)," 2017. [Online]. Available: https://tools.ietf.org/id/draft-ietf-lwig-tcp-constrained-node-networks-01.html#RFC6298

[15] http archive, "Interesting stats," Jan. 2018. [Online]. Available: http://httparchive.org/interesting.php

[16] M. Iglesias-Urkia, A. Orive, M. Barcelo, A. Moran, J. Bilbao, and A. Urbieta, "Towards a lightweight protocol for industry 4.0: An implementation based benchmark," in *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, May 2017, pp. 1–6.