Strato Applicativo
HTTP2 e HTTP3
approfondimenti

Reti di Calcolatori
AA. 2023-2024

Docente: Federica Paganelli
Dipartimento di Informatica
federica.paganelli@unipi.it

Approfondimento – HTTP Page Load Time

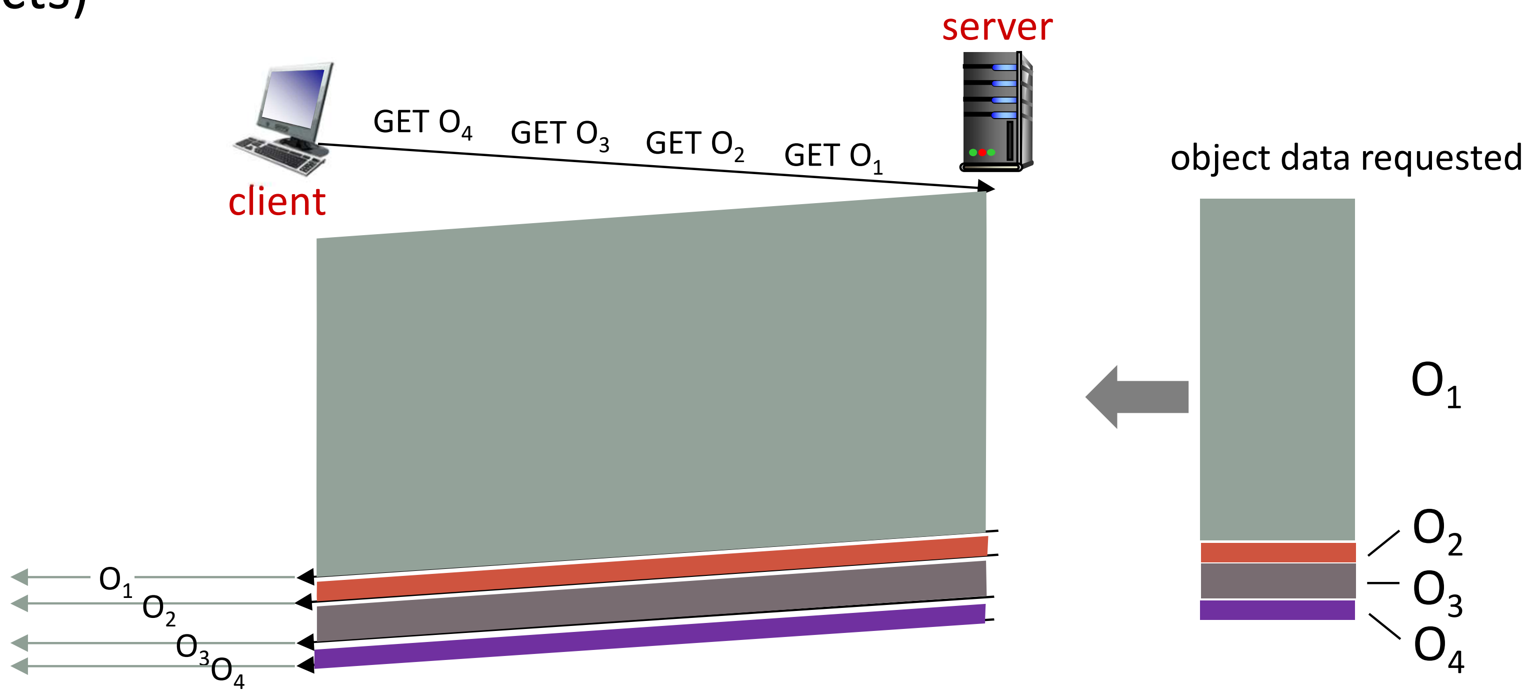
Come migliorare il tempo di caricamento di una pagina agendo a livello di protocollo?

- Con HTTP/1.1:
  - più GET in pipeline su una singola connessione TCP
    - il server risponde in ordine (FCFS: first-come-first-served scheduling), un oggetto piccolo potrebbe dover attendere la trasmissione (Head of Line Blocking - HOL) dietro ad oggetti di grandi dimensioni
    - il ripristino delle perdite (ritrasmissione di segmenti TCP persi) introduce ulteriore ritardo la trasmissione dell'oggetto
  - Altra soluzione con HTTP1.1: i browser instaurano più connessioni TCP in parallelo (tipicamente con un valore massimo di 6 connessioni contemporanee)
    - Comportamento non equo
    - Poco efficace controllo di congestione

HTTP/1.1 HOL blocking

Head of Line blocking: una risposta può bloccare l'invio dei contenuti successivi

HTTP 1.1: client requests 1 large object (e.g., video file, and 3 smaller objects)



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

Approfondimento – HTTP Page Load Time

Come migliorare il tempo di caricamento di una pagina agendo a livello di protocollo?

- HTTP/2 (RFC 7540)

  https://tools.ietf.org/html/rfc7540

- HTTP/3 (RFC 9114)

  https://datatracker.ietf.org/doc/html/rfc9114

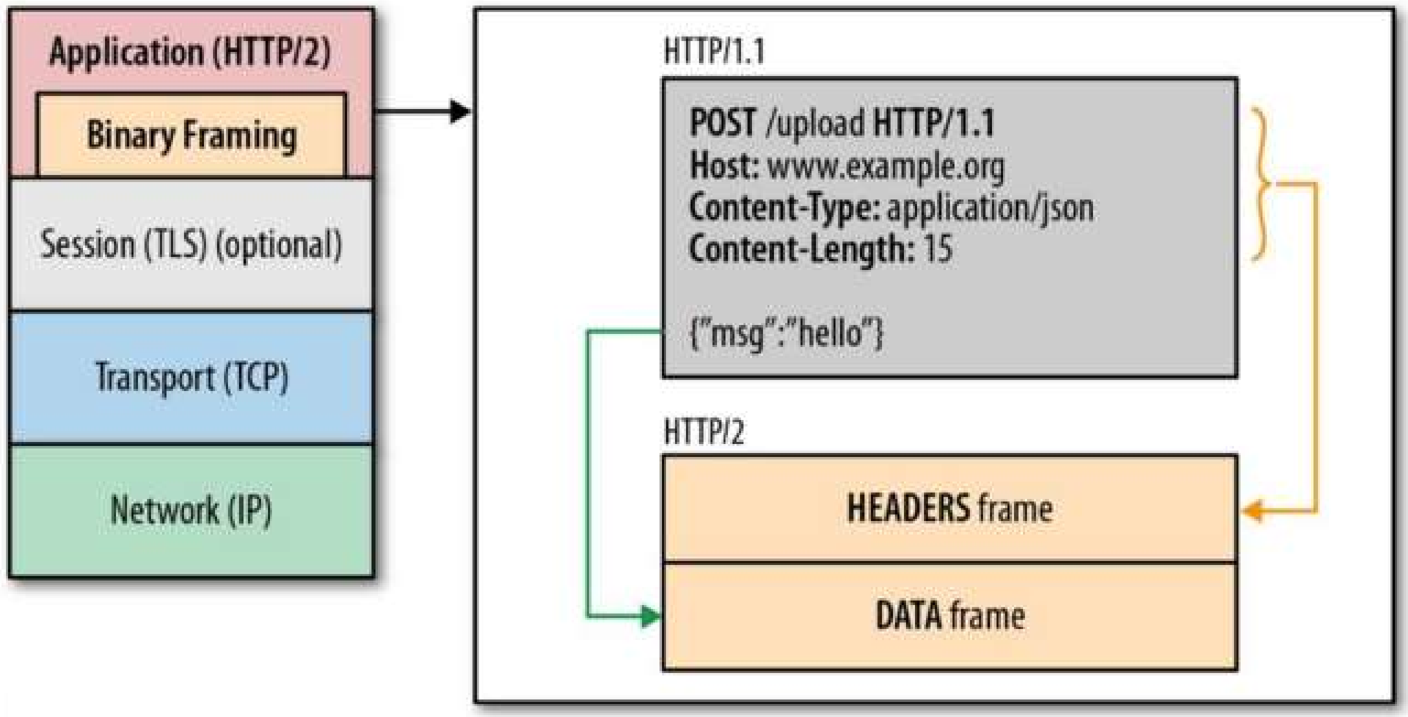
HTTP/2

- maggiore flessibilità lato server nell'invio di oggetti al client
- metodi, codici di stato, la maggior parte dei campi di intestazione invariati rispetto a HTTP 1.1
- ordine di trasmissione degli oggetti richiesti in base alla priorità dell'oggetto specificata dal client (non necessariamente FCFS)
- inviare oggetti non richiesti al cliente (Push)
- dividere gli oggetti in frame e scheduling dei frame per mitigare Head of Line (HoL) blocking

HTTP/2

1. Multiplexing delle richieste su una unica connessione TCP

- **Frame:** l'unità di comunicazione in HTTP/2. Una sequenza di frame costituisce un messaggio HTTP (es. un frame può essere un HEADER frame, DATA frame..)



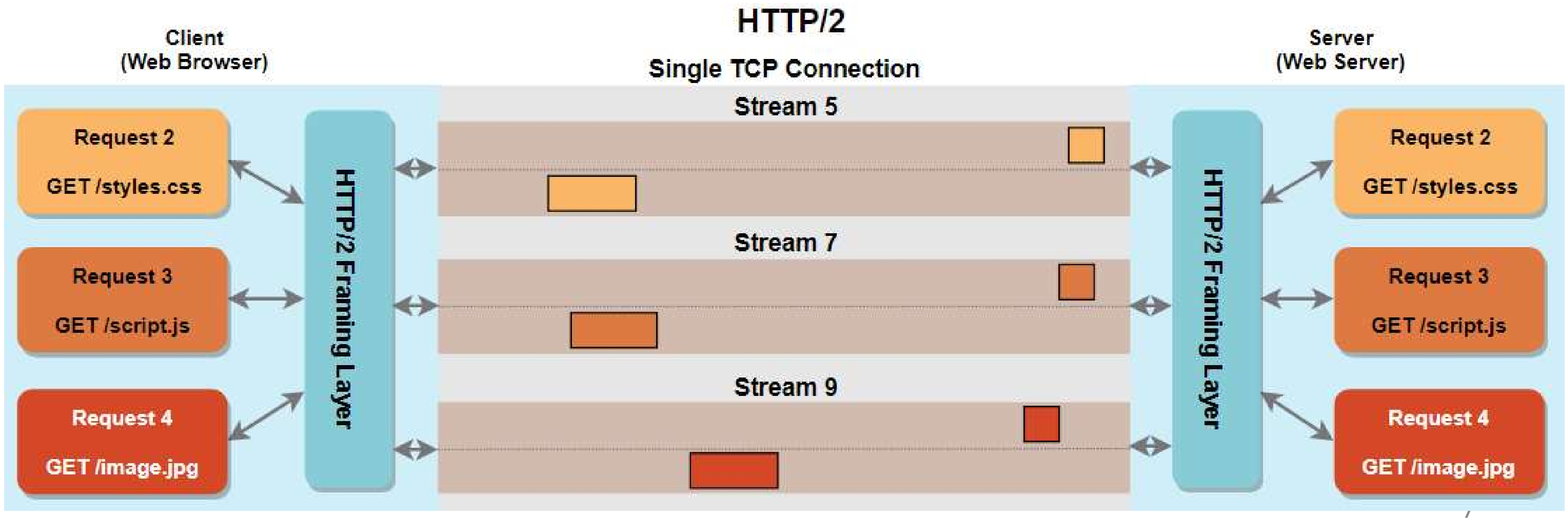
```
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15

{"msg":"hello"}
```

Grigorik et al., 2014

https://blog.restcase.com/http2-benefits-for-rest-apis/

HTTP/2

- **Stream:** un flusso bidirezionale di frame all'interno di un'unica connessione TCP, rappresenta una comunicazione richiesta-risposta
- Mediante l'astrazione degli stream è possibile effettuare il **multiplexing delle richieste** (più stream sulla stessa connessione TCP)



https://dzone.com/articles/understanding-http2

HTTP/2

2. Definizione delle priorità

E’ possibile associare ad ogni stream:

- Un **peso** che ne indica la priorità
- Una **dipendenza** verso altri stream

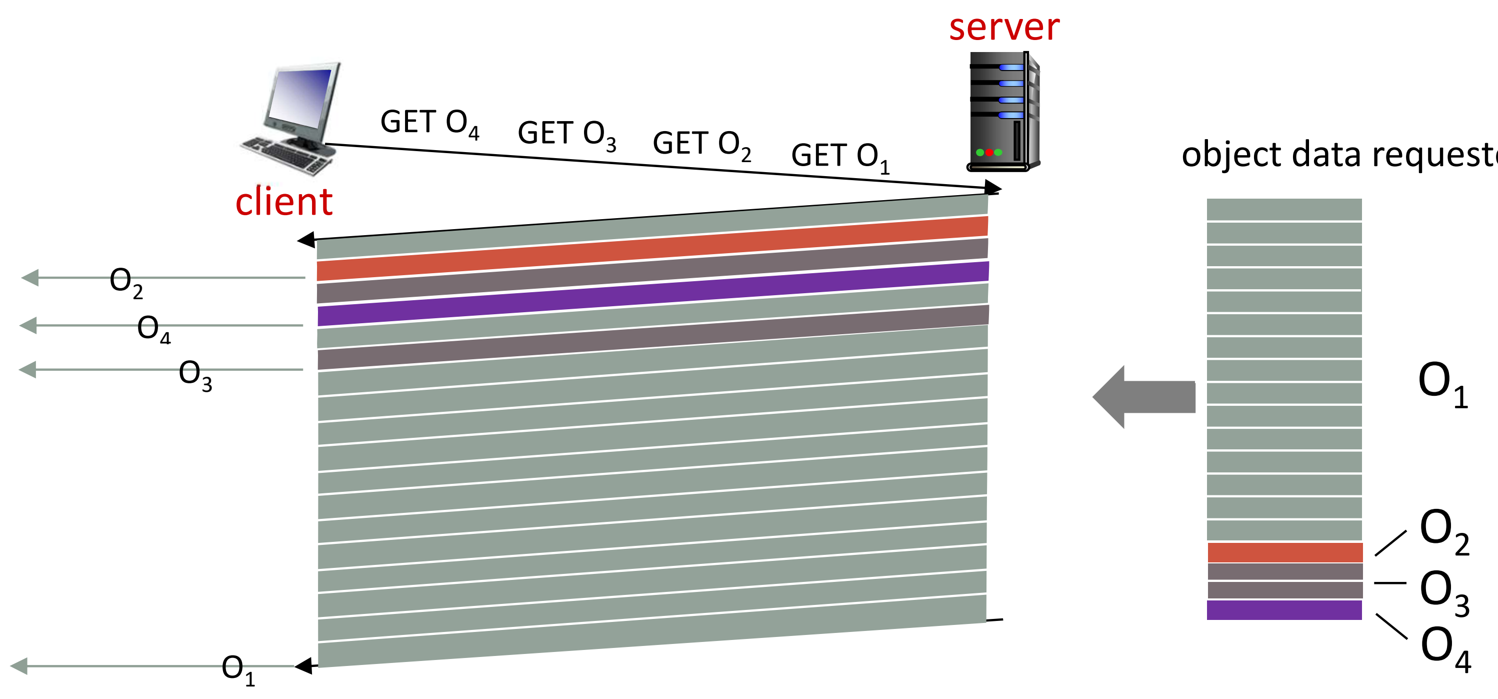
3. Compressione delle intestazioni

4. Server Push: Il meccanismo di **Server Push**, permette al server di inviare risorse aggiuntive per una singola richiesta da parte del client

Grigorik et al., 2014

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



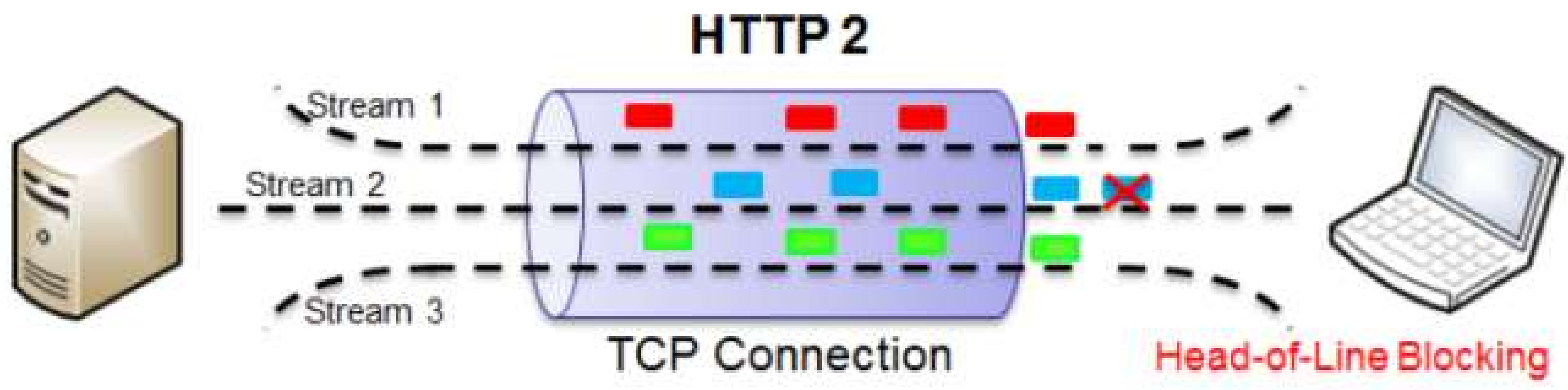
O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

HTTP/2

- **HTTP/2 risolve il problema di uso inefficiente di una connessione TCP**
  - **In particolare per pagina con tante risorse**
  - **Vedi demo** **https://http2.akamai.com/demo**

    https://http1.akamai.com/demo/h1_demo_frame.html

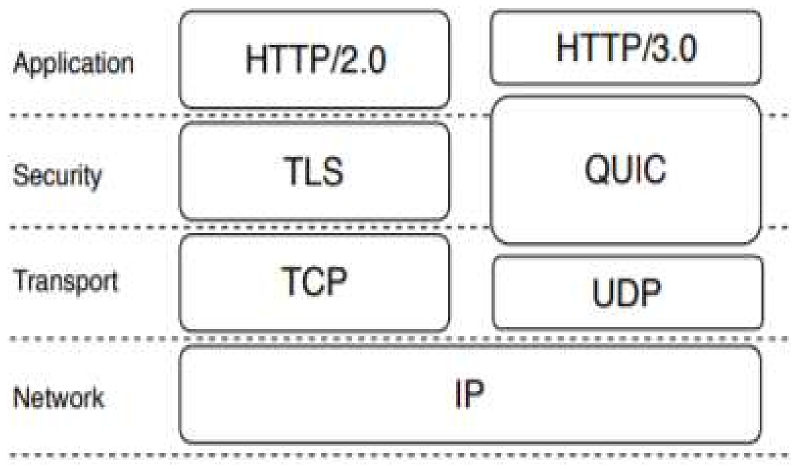
    https://http2.akamai.com/demo/h2_demo_frame.html
  - **ma…**
- LIMITE: Non risolve completamente il problema di **Head Of Line Blocking.** Questo perché TCP non «vede» gli stream. Ad esempio, una perdita (3 ack/timeout) provoca lo stallo della connessione (e quindi tutti gli stream)



Wang et al. 2014

HTTP/3

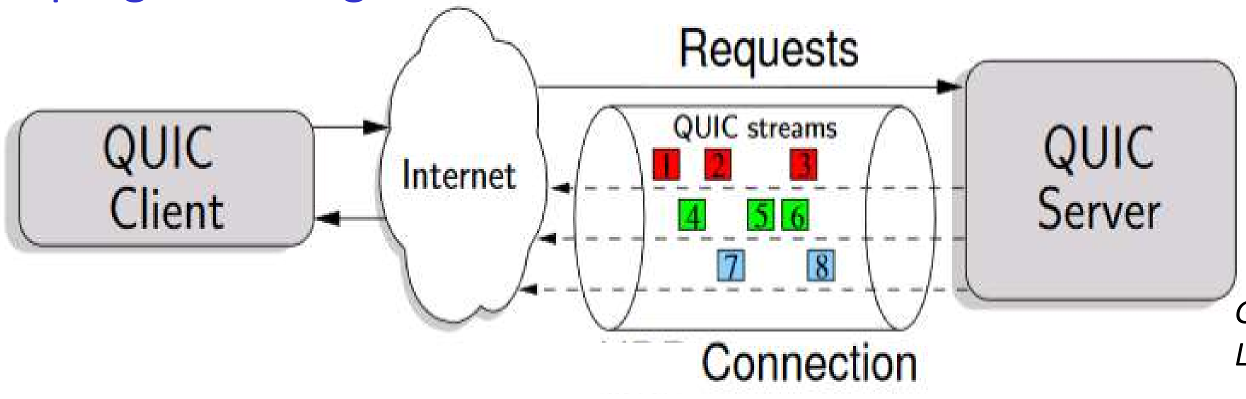
- Evoluzione di HTTP/2 che usa i servizi di QUIC (UDP)
- Mantiene la semantica di HTTP/2
- aggiunge sicurezza, controllo degli errori e controllo della congestione sopra UDP



Gratzer, F. 2016
Langley et al. 2017

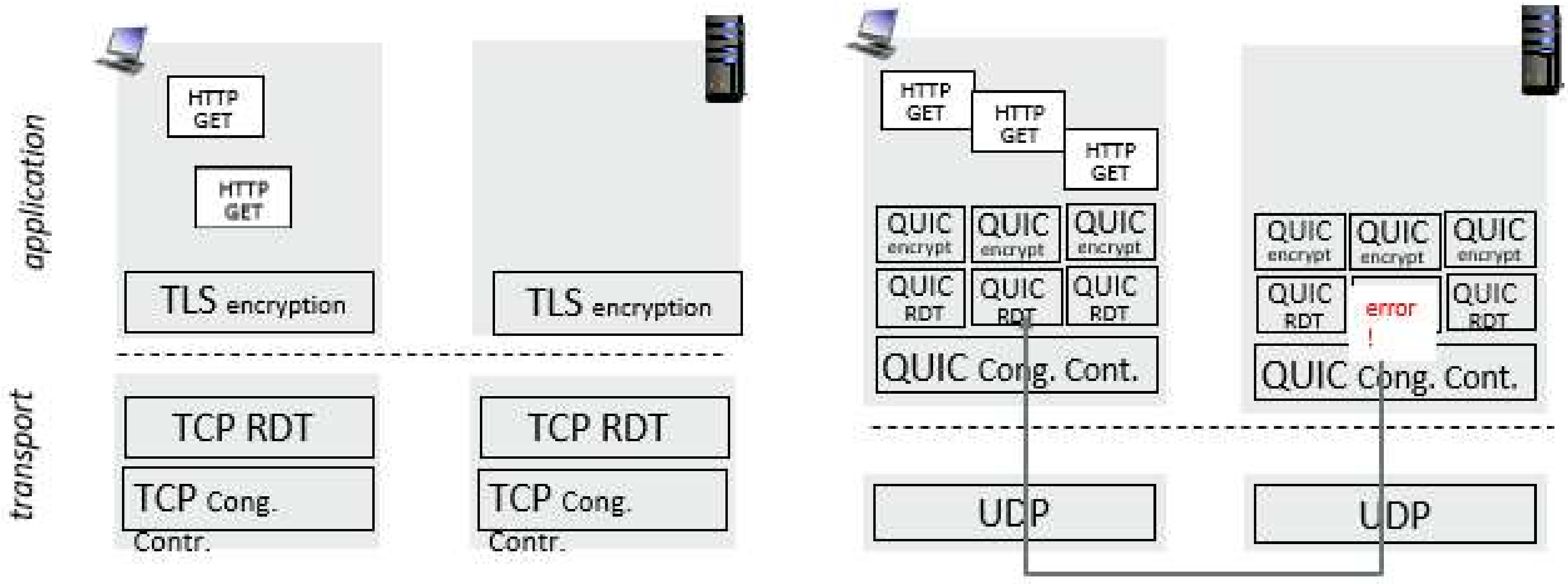
QUIC

- https://datatracker.ietf.org/doc/html/rfc9000
- Protocollo di trasporto che sfrutta UDP ed è orientato alla connessione
- Aggiunge:
  - Controllo del flusso e della congestione
  - Rilevazione delle perdite e ritrasmissione
- IMPORTANTE
- più "flussi" a livello di applicazione multiplexati su una singola connessione QUIC
- Astrazione dello stream gestito a livello di trasporto
- Trasferimento dati affidabile sul singolo flusso
- Gli stream sono indipendenti tra loro, la perdita o rallentamento di uno stream non influisce sul progredire degli altri streams



Gratzer, F. 2016
Langley et al. 2017

QUIC: streams: parallelism, no HOL blocking

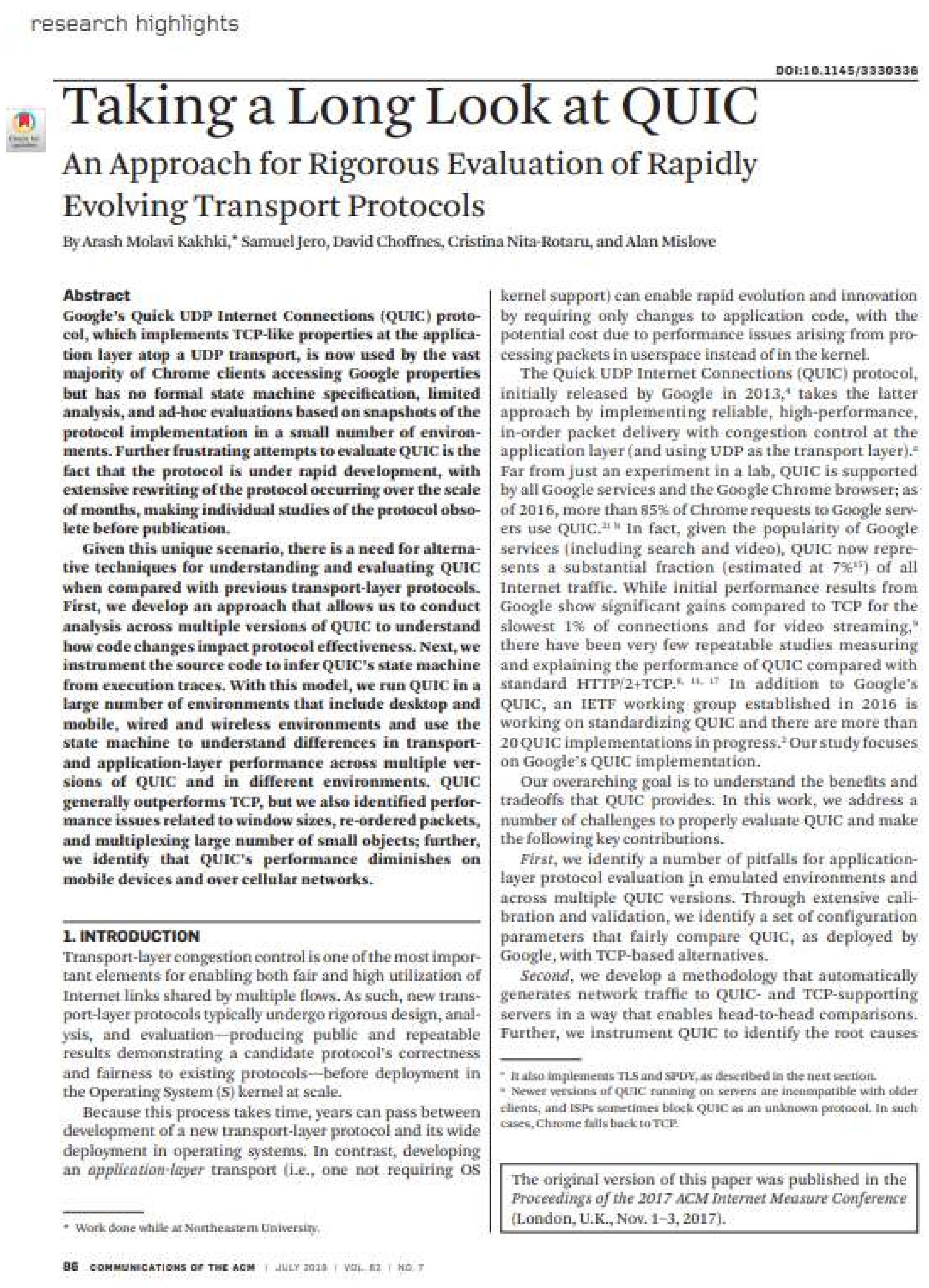


(a) HTTP 1.1

(b) HTTP/2 with QUIC: no HOL blocking

TCP vs QUIC

- Kakhki, A.M., Jero, S., Choffnes, D., Nita-Rotaru, C. and Mislove, A., 2019. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols. *Communications of the ACM*, *62*(7), pp.86-94. https://dl.acm.org/doi/pdf/10.1145/3330336



DOI:10.1145/3330336

Taking a Long Look at QUIC

An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols

By Arash Molavi Kakhki,* Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove

Abstract

Google's Quick UDP Internet Connections (QUIC) protocol, which implements TCP-like properties at the application layer atop a UDP transport, is now used by the vast majority of Chrome clients accessing Google properties but has no formal state machine specification, limited analysis, and ad-hoc evaluations based on snapshots of the protocol implementation in a small number of environments. Further frustrating attempts to evaluate QUIC is the fact that the protocol is under rapid development, with extensive rewriting of the protocol occurring over the scale of months, making individual studies of the protocol obsolete before publication.

Given this unique scenario, there is a need for alternative techniques for understanding and evaluating QUIC when compared with previous transport-layer protocols. First, we develop an approach that allows us to conduct analysis across multiple versions of QUIC to understand how code changes impact protocol effectiveness. Next, we instrument the source code to infer QUIC's state machine from execution traces. With this model, we run QUIC in a large number of environments that include desktop and mobile, wired and wireless environments and use the state machine to understand differences in transport- and application-layer performance across multiple versions of QUIC and in different environments. QUIC generally outperforms TCP, but we also identified performance issues related to window sizes, re-ordered packets, and multiplexing large number of small objects; further, we identify that QUIC's performance diminishes on mobile devices and over cellular networks.

1. INTRODUCTION

Transport-layer congestion control is one of the most important elements for enabling both fair and high utilization of Internet links shared by multiple flows. As such, new transport-layer protocols typically undergo rigorous design, analysis, and evaluation—producing public and repeatable results demonstrating a candidate protocol's correctness and fairness to existing protocols—before deployment in the Operating System (S) kernel at scale.

Because this process takes time, years can pass between development of a new transport-layer protocol and its wide deployment in operating systems. In contrast, developing an *application-layer* transport (i.e., one not requiring OS kernel support) can enable rapid evolution and innovation by requiring only changes to application code, with the potential cost due to performance issues arising from processing packets in userspace instead of in the kernel.

The Quick UDP Internet Connections (QUIC) protocol, initially released by Google in 2013,[4] takes the latter approach by implementing reliable, high-performance, in-order packet delivery with congestion control at the application layer (and using UDP as the transport layer).[a] Far from just an experiment in a lab, QUIC is supported by all Google services and the Google Chrome browser; as of 2016, more than 85% of Chrome requests to Google servers use QUIC.[21 b] In fact, given the popularity of Google services (including search and video), QUIC now represents a substantial fraction (estimated at 7%[15]) of all Internet traffic. While initial performance results from Google show significant gains compared to TCP for the slowest 1% of connections and for video streaming,[9] there have been very few repeatable studies measuring and explaining the performance of QUIC compared with standard HTTP/2+TCP.[8, 11, 17] In addition to Google's QUIC, an IETF working group established in 2016 is working on standardizing QUIC and there are more than 20 QUIC implementations in progress.[2] Our study focuses on Google's QUIC implementation.

Our overarching goal is to understand the benefits and tradeoffs that QUIC provides. In this work, we address a number of challenges to properly evaluate QUIC and make the following key contributions.

First, we identify a number of pitfalls for application-layer protocol evaluation in emulated environments and across multiple QUIC versions. Through extensive calibration and validation, we identify a set of configuration parameters that fairly compare QUIC, as deployed by Google, with TCP-based alternatives.

Second, we develop a methodology that automatically generates network traffic to QUIC- and TCP-supporting servers in a way that enables head-to-head comparisons. Further, we instrument QUIC to identify the root causes

[a] It also implements TLS and SPDY, as described in the next section.

[b] Newer versions of QUIC running on servers are incompatible with older clients, and ISPs sometimes block QUIC as an unknown protocol. In such cases, Chrome falls back to TCP.

The original version of this paper was published in the *Proceedings of the 2017 ACM Internet Measure Conference* (London, U.K., Nov. 1–3, 2017).

* Work done while at Northeastern University.

TCP vs QUIC

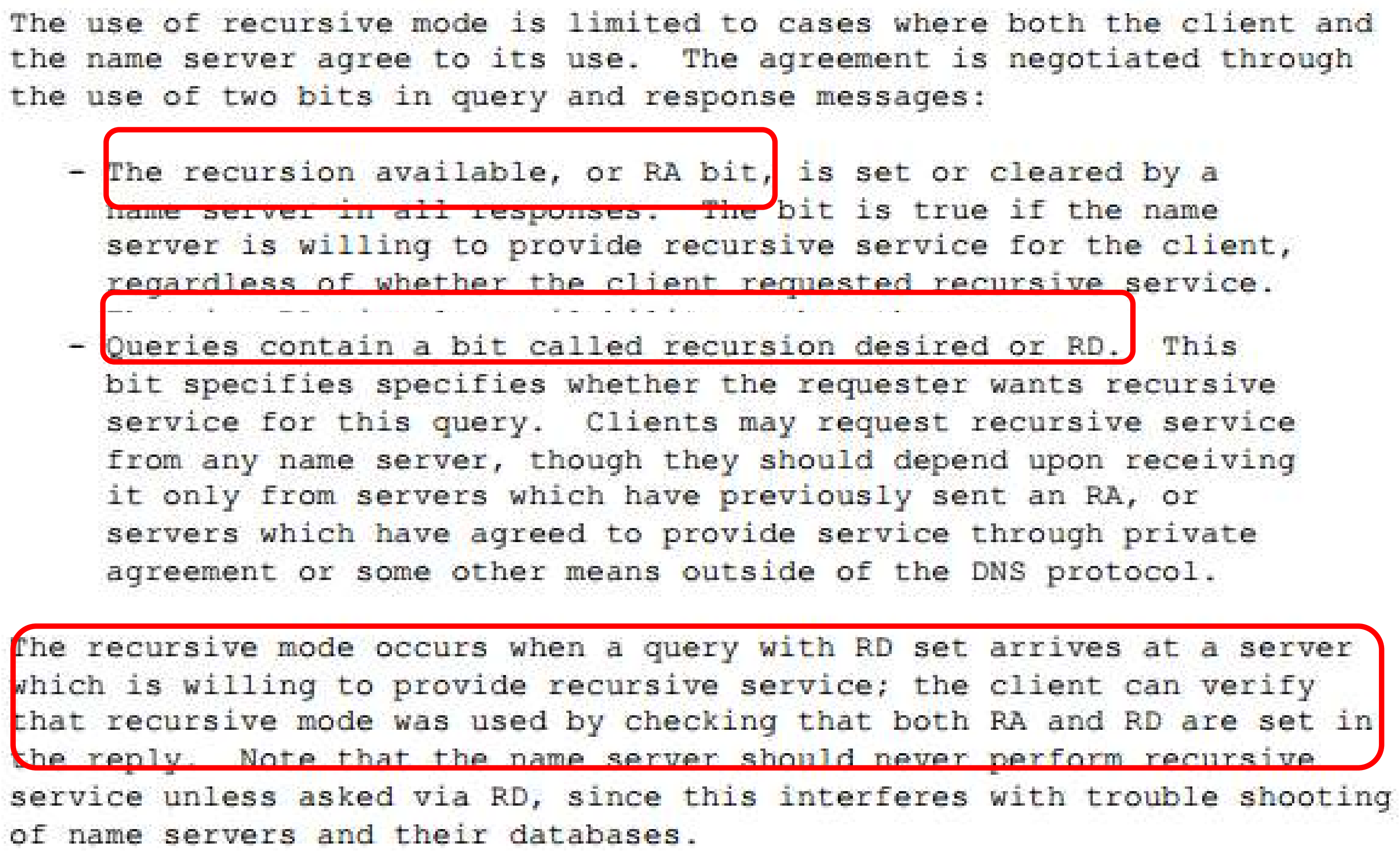
- Kakhki, A.M., Jero, S., Choffnes, D., Nita-Rotaru, C. and Mislove, A., 2019. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols. *Communications of the ACM*, *62*(7), pp.86-94. https://dl.acm.org/doi/pdf/10.1145/3330336

Risultati principali:

- QUIC generally outperforms TCP
- but performance issues have been identified related to re-ordered packets, multiplexing large number of small objects, etc.
- QUIC's performance diminishes on mobile devices and over cellular networks

Approfondimento DNS

Determinare, analizzando la RFC 1034, in che modo viene stabilito se una query DNS viene gestita in modo ricorsivo o meno.



The use of recursive mode is limited to cases where both the client and the name server agree to its use. The agreement is negotiated through the use of two bits in query and response messages:

- The recursion available, or RA bit, is set or cleared by a name server in all responses. The bit is true if the name server is willing to provide recursive service for the client, regardless of whether the client requested recursive service. [illegible]

- Queries contain a bit called recursion desired or RD. This bit specifies specifies whether the requester wants recursive service for this query. Clients may request recursive service from any name server, though they should depend upon receiving it only from servers which have previously sent an RA, or servers which have agreed to provide service through private agreement or some other means outside of the DNS protocol.

The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive mode was used by checking that both RA and RD are set in the reply. Note that the name server should never perform recursive service unless asked via RD, since this interferes with trouble shooting of name servers and their databases.