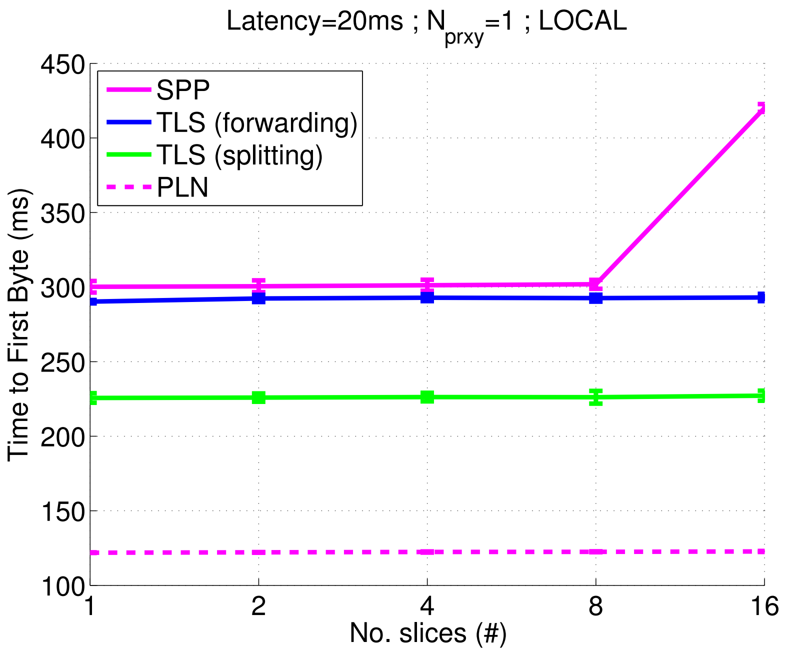
1 Experiments

We differentiate between two main environments: *synthetic* and *realistic*.
Synthetic environment: It consists of a single machine where we run concurrently one or several instances of our applications (client, mbox and server). In this environment we control bandwidth, delay and losses using Linux TC [**?**]. Also, we can easily run as many instances of each application as we want without requiring any further machine. If not specified otherwise, each experiment consists of 50 runs for which we report average and standard deviation by mean of error bars. We also assume that network links have a speed capped at 8 Mbps.
Realistic environment: It consists of three different machines on which we run respectively our client, mbox and server. The client runs on a desktop machine in our lab with both wired and 3G access. The mbox and the server run on two different Amazon machine we start on different locations.

TODO: (1) Any benefit of running VMs? (2) Add control for losses? (3) 3G experiments (4) Amazon locations



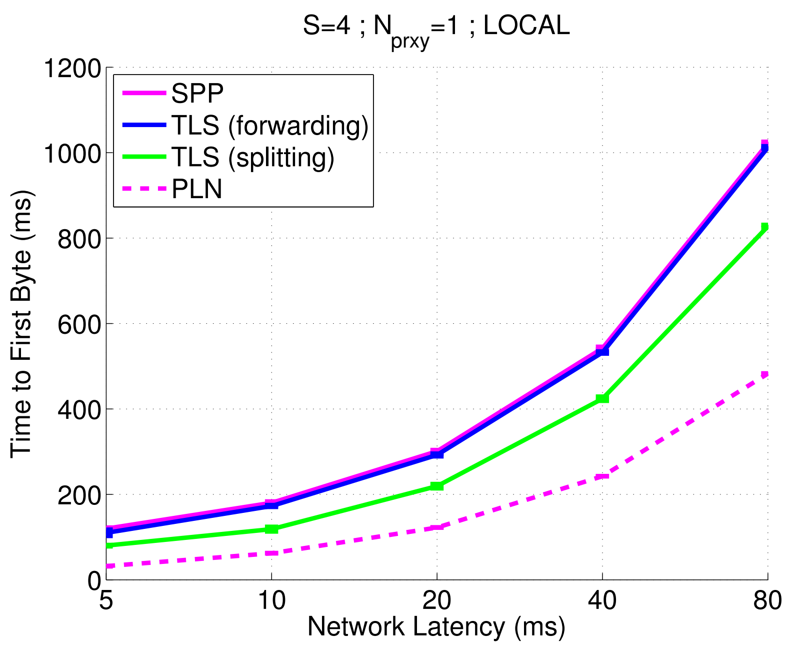
1.1 Controlled

Figure 1.1 shows the time to first byte as a function of the number of slices S and mode of operation. In this scenario, we run one client, mbox and server, which are connected through an 8 Mbs link with 20 ms network delay, *i.e.,* 80 ms RTT between client and server. We consider a worst case scenario for `TruMP` where the mbox has read/write access to each slice, independently of the number of slices. Note that, unless otherwise specified, in the coming scenarios we will always assume each mbox has full read/write access to each slice.

In Figure 1.1 `NoEncrypt`, or no encryption, shows the baseline of our experiments, with a time to first byte of about 120 ms or 1.5 RTT. We now add focus on the modes dealing with encrypted traffic. Compared to today (`E2E-TLS`), `TruMP` only add a negligible delay of 10 ms when S is between 1 and 8 slices. With 16 slices, the time to first byte with `TruMP` grows by about 50%. This is due to the Nagle algorithm (...). `SplitTLS`, overall, saves about 1 RTT compared to both `TruMP` and `NoEncrypt`: this is due to the fact that the two TLS sessions (client-mbox and mbox-server) carry on in parallel. Note that we could also implement `TruMP` in a split fashion and enjoy a similar performance boost, but this would indeed violate our end-to-end principle.

TODO: Q: why again is `NoEncrypt` 1.5 RTT?

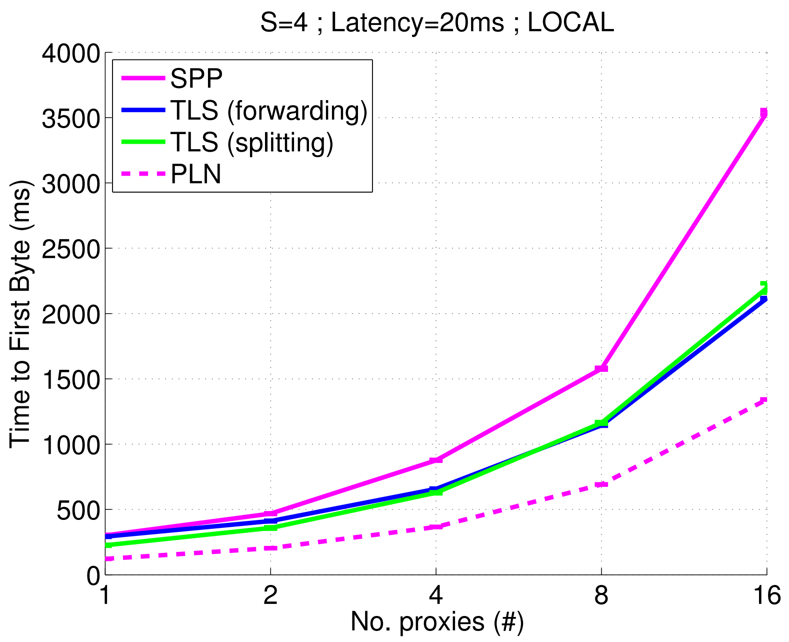
TODO: Run experiment for Nagel algorithm (add support to client) and explain results



We now consider a scenario with fixed number of slices and variable network delay. We set S equal to 4, assuming two slices are used for user requests (header and body) and two slices are used for server responses (header and body). We then vary the network delay between a minimum of 5 and up to 80 ms. Since we use a single mbox instance, this means an RTT between client and server that varies between 20 and 320 ms. Overall, results above are confirmed.

TODO: Discuss strategy at server (uni, cs). Check what we ran with.

TODO: Add x axis on top with RTT measure



We now consider a scenario with variable number of mboxes. We set S equal to 4, the network delay to 20 ms, and vary the number of mboxes N between 1 and 16.

TODO: Q: why splitting becomes worst than fwd with 16 mboxes? TODO: run experiment with 0 middleboxes

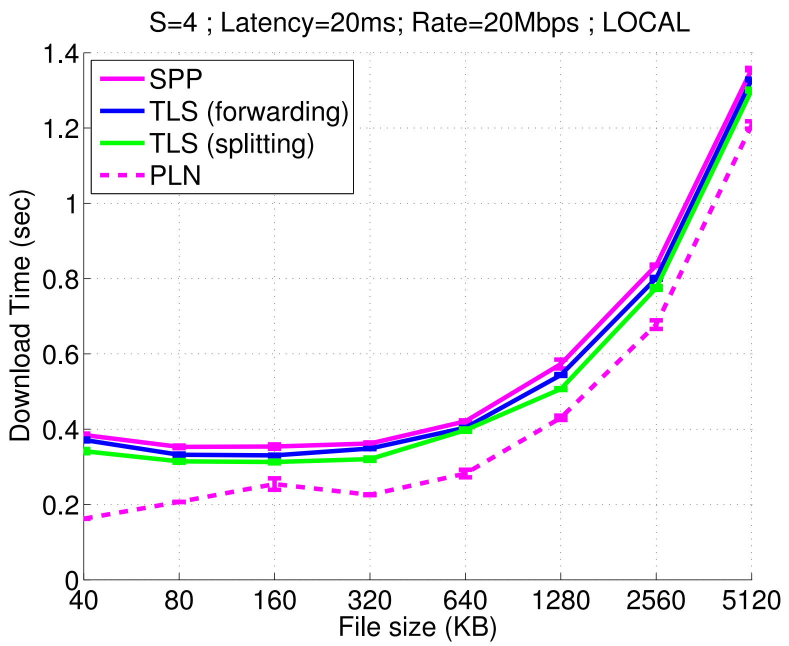
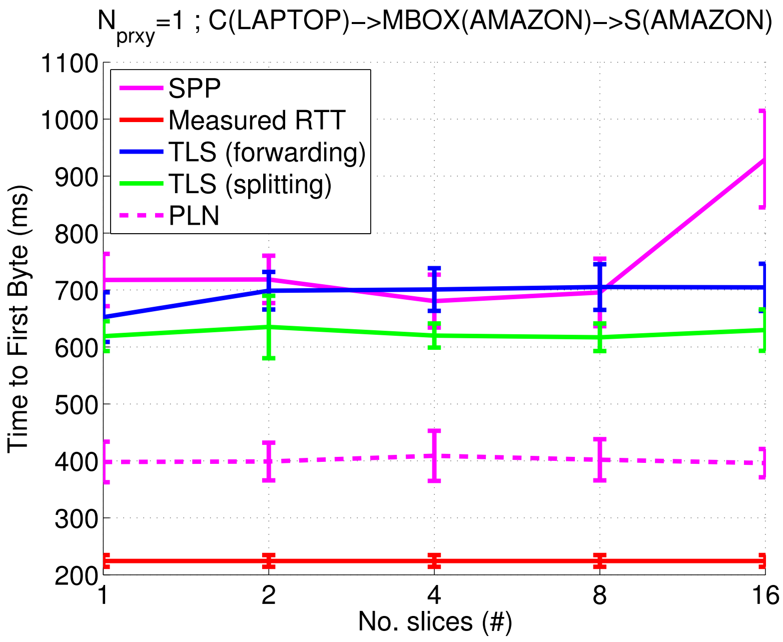


Figure 1.1 shows the download time as a function of the file transferred for each mode of operation described above. In this scenario, we set S equal to 4, the network delay to 20 ms, and N equal to one. As expected from the time to first byte analysis, `TruMP` only minimally increases the download time compared to `E2E-TLS`: for example, for a file transfer of 1280 KB we measure a download time of 570 and 540 ms for `TruMP` and `E2E-TLS`, *i.e.,* an increase of only 5%. While difference in download time between `TruMP` and `SplitTLS` is more prominent, it becomes negligible at the size of the transferred file grows (...)

TODO: Transform the figure as percentace of increase time versus the fastest mode (no encryption)

TODO: Q: Do a plot with scenarios where we play with file size and link speed. Then in final scenario we use a realistic one (Amazon)

TODO: Q: Weird behavior for small sizes? Maybe due to small number of repetitions? Try increase to 50 when time (now only 10 rep)

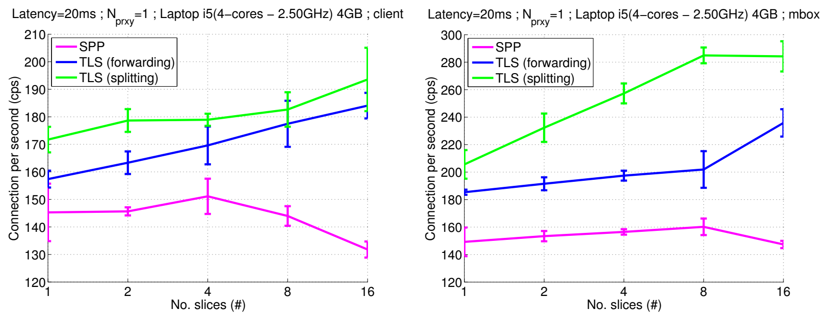


1.2 Realistic Environment

We start by considering a scenario where the client, located in Barcelona (Spain), is connected to the Internet with a fiber connection (avg measured speed of 45 mbps). The mbox runs on an Amazon EC2 instance located in Ireland and the server runs on an Amazon EC2 instance located is California.

Figure 1.2 shows the time to first byte as a function of the number of slices S and mode of operation in a realistic scenario. The figure also shows the measured RTT between client and server. Accordingly, results from controlled environment are verified.

TODO: measure speed from TID



(a) Client.

(b) Mbox.

Figure 1: Connection per second.

1.3 Scalability

We now aim to understand the impact of the additional complexity of `TruMP` on the number of connections per second that a server (and a mbox) can sustain. To do so, we extend `s_time`, the measuring application contained within the OpenSSL package, to support `TruMP`. Overall, this required about X line changes.

`S_time` infers the number of connections per second a client can launch without invoking the server through concurrent connections. Instead, it opens sequential connection, measure the CPU time consumed in a given time-frame and then derive theoretical number of connections per second. For instance, if we made 100 connection in 0.5 CPU seconds, then we derive that 200 connections per second are possible. We leverage the same rationale to measure the number of connections per second that a mbox and server can sustain.

TODO: Derive line changes

Figure 1 compares the number of connections per second measured respectively by client, mbox and server on an Intel i5 (4-cores - 2.50GHz) with 4 GB of memory.

TODO: Due to an error in the script, server results are missing. REDO

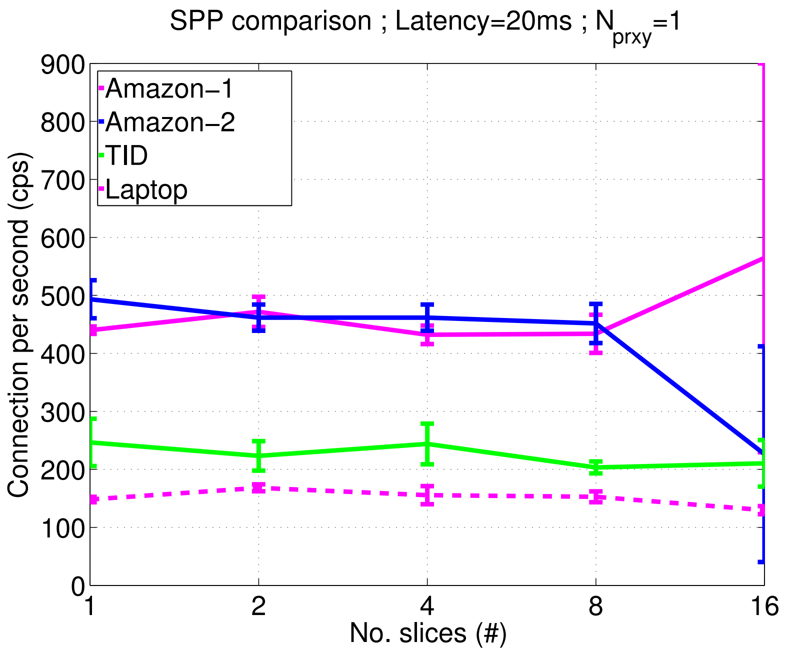


Figure 1.3 compares the performance of `TruMP` on different hardware.
TODO: REDO with new code