Performance of Different Proxy Concepts in UMTS Networks

Marc C. Necker[1], Michael Scharf[1], and Andreas Weber[2]

[1] Institute of Communication Networks and Computer Engineering
University of Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart
{necker,scharf}@ikr.uni-stuttgart.de

[2] Alcatel SEL AG, Research and Innovation
Lorenzstr. 10, D-70435 Stuttgart
Andreas.Weber@alcatel.de

Abstract It is well known that the large round trip time and the highly variable delay in a cellular network may degrade the performance of TCP. Many concepts have been proposed to improve this situation, including performance enhancing proxies (PEPs). One important class of PEPs are split connection proxies, which terminate a connection from a server in the Internet in a host close to the Radio Access Network (RAN) and establish a second connection towards the mobile User Equipment (UE). This connection splitting can be done either purely on the transport layer (TCP proxy) or on the application layer (HTTP proxy in the case of web traffic). While it is clear that an application layer proxy also infers the splitting of an underlying transport layer connection, the performance of applications may be essentially different for both approaches. In this paper, we first study the general impact of a split connection proxy on TCP bulk data transfer. We then focus on the case of web traffic and investigate the TCP connection behavior of the Mozilla web browser. Based on this, we study the performance of TCP and HTTP proxies in UMTS networks under different scenarios and for different HTTP configurations.

1 Introduction

The access to the World Wide Web is one of the most important applications in today's Internet. Since UMTS networks provide much higher data rates compared to GSM-based cellular networks, surfing the Web is expected to be a very promising service. The characteristics of mobile networks differ from fixed networks due to the error-prone radio channel and terminal mobility. Therefore, it is important to understand how wireless web access can be improved. This usually requires cross-layer optimization of all involved protocol layers, in particular including the transport layer.

There are two main approaches to address this issue: First, the performance can be optimized end-to-end by using well-tuned protocol mechanisms, i.e. by using optimized TCP versions. This typically requires modifications in the protocol stacks in the end-systems and is thus difficult to deploy in the global Internet.

Alternatively, a performance enhancing proxy (PEP) [1] can be used between the radio access network and the Internet. These proxies can operate at different protocol layers. In the case of web traffic, the proxy can either operate at the transport layer (TCP proxy) or at the application layer (HTTP proxy). While HTTP proxies are already present in todays networks, TCP proxies have not been deployed yet. However, a number of approaches have been presented in literature, such as *Indirect TCP* [2] and *Multiple TCP* [3].

In [18], Chakravorty et al. evaluated the WWW performance in GPRS networks by means of measurements. A comprehensive discussion of proxies in general, and how TCP proxies can improve the throughput of a TCP download session can be found in [5]. After a brief study of the impact of PEPs on TCP bulk data transfer, we focus on the WWW as an application in this paper. Consequently, we do not limit ourselves to the transport layer only. Instead, we explicitly take into account the behavior of the WWW application, i.e. the way TCP connections are handled by web browsers. In particular, we study the behavior of the Mozilla web browser with different HTTP configurations and evaluate the performance gain of TCP and HTTP proxies compared to a non-proxied scenario by means of emulation.

This paper is organized as follows. In section 2, we give a brief survey of PEPs. In section 3, we describe the scenario and the emulation environment used for our studies. Next, we present and discuss the results for the case of FTP and WWW traffic in section 4 and section 5, respectively. Finally, we conclude the paper in section 6.

2 Performance Enhancing Proxies in Mobile Networks

The use of PEPs in mobile networks has been studied extensively. Most of the research has been motivated by the fact that TCP does not perform well in the presence of non congestion related packet loss. On the other hand, e.g. in UMTS, for PS traffic over DCH, the RLC layer can be configured to provide excellent reliability by using Forward Error Correction (FEC) and Automatic Repeat reQuest (ARQ). Furthermore, the UTRAN ensures in-order delivery. As a consequence, well-configured TCP connections [6] hardly benefit from PEPs which perform local error recovery and in-order delivery [5]. However, UTRAN error protection comes at the cost of higher latency and delay jitter [7].

High latencies and high bandwidth delay products are a challenge for TCP. First, the connection establishment takes longer. Second, if the bandwidth delay product is greater than the transmitter window size, the pipe between server and client cannot be fully utilized by the TCP connection. The transmitter window size is reduced at the start of a TCP connection (Slow Start), but also after a Retransmission Timeout (RTO), which causes the system to perform Slow Start followed by Congestion Avoidance. Furthermore, the transmitter window size may never exceed the advertised window size of the receiver which may be reduced, e.g. due to buffer problems in the receiver. Beside these issues, round trip delay jitters in the order of seconds can trigger spurious TCP timeouts, resulting in unnecessarily retransmitted data [8]. In the following, we will give a brief overview how these problems can be mitigated by different proxy concepts [1].

2.1 Protocol Helpers

Protocol helpers are able to add, manipulate, resort, duplicate, drop or delay messages. This includes data messages as well as acknowledgments. However, they neither terminate a connection, nor modify user data. The classical example for a protocol helper is the Berkeley Snoop Protocol [9], which buffers unacknowledged TCP segments and retransmits them locally in the RAN in case of a packet loss. Moreover, acknowledgments are filtered in order to hide packet losses from the TCP sender. However, e.g. for UMTS DCH channels, such a local recovery is done much more efficiently by the UMTS RLC layer.

Protocol helpers have also been proposed to improve TCP performance in the presence of spurious timeouts, either by filtering redundant segments [10], by buffering of acknowledgments [11] or by manipulating the receiver advertised window [12]. However, none of these approaches can mitigate problems caused by high bandwidth delay products.

2.2 TCP proxies

A TCP proxy is an entity which, from the perspective of an Internet server, is located before the RAN. It splits the TCP connections into one connection in the fixed network part and one connection in the mobile network part. TCP proxies are a well-known approach to improve TCP performance in wireless networks [2, 3] and have extensively been studied in literature. For instance, Meyer et al. [5] showed that a TCP proxy can significantly improve TCP throughput, especially in case of high data rates (i.e. 384kBit/s in UMTS).

TCP proxies shield the mobile network from potential problems in the Internet. Usually, the transmission delay in the fixed network part is much smaller compared to the delay in the mobile network. Hence, a TCP connection in the fixed network part recovers much faster from packet losses. Another advantage is the high flexibility, since TCP proxies can be modified without great effort. In particular, the PEP's TCP sender directed towards the wireless link can use optimized algorithms, which might not be suitable for the worldwide Internet.

2.3 HTTP proxies

HTTP proxies are well understood and commonly used in wireline networks. Typically, the user can decide whether a proxy shall be used by configuring the web browser accordingly. As with all application layer proxies, HTTP proxies split the underlying transport layer connections. Additionally, they cache frequently accessed data and thus may reduce page loading times[3]. The proxy reduces the number of DNS lookups over the wireless link as it can perform

[3] Note that this might not be possible for dynamic pages

these lookups on behalf of the web browser. If the persistence feature in HTTP is activated, it may also reduce the number of TCP connections, since the user equipment usually accesses the same proxy for the duration of a session. It can then maintain one or several persistent TCP connections to that proxy, as compared to several shorter lasting connections when connecting to different servers in the Internet. This highly improves the performance as the overhead of connection setup is removed and the connections are less likely to be in slow start.

2.4 Implications of Using PEPs

All PEPs violate the end-to-end argument and the protocol layering, two fundamental architectural principles of the Internet [1]. They require additional processing and storage capacity and have limited scalability. Furthermore, a proxy is an additional error source, and end-systems might not be able to correct errors occurring within a proxy. The location of a PEP within the network architecture has to be chosen very thoroughly since it might be necessary to transfer state information if the route from the user equipment to the Internet changes. Finally, proxies are not compatible to encryption and digital signature on the IP-layer, i.e. IPsec. This implies that a PEP might also prevent the usage of Mobile IP.

Border et al. disadvise proxies that automatically intervene in all connections [1]. Instead, it is recommended that end users should be informed about the presence of a proxy and should have the choice whether to use it or not. Such a procedure would favor the deployment of application-specific proxies.

3 Scenario and Simulation/Emulation Environment

3.1 Network Scenario and Simulation Environment

The basic scenario is shown in Fig. 1 (top). We consider a single-cell environment, where the User Equipment (UE) on the left side connects to the Node B via a 256kBit/s dedicated channel (DCH) in the uplink and the downlink direction. The Node B is connected to the Radio Network Controller (RNC), which itself is connected to the Internet via the 3G-SGSN and 3G-GGSN of the cellular system's core network. Finally, the UE establishes the data connection with a web server in the Internet. The Internet and core network were assumed to introduce a constant delay T_{INet} and randomly lose IP packets with a probability of P_{loss}. The UTRAN was modeled in detail with all its relevant protocols and parametrized according to the optimal parameters found in [7]. It was implemented using the event-driven simulation library *IKR SimLib* [13]. The loss probability for a MAC frame on the air interface was set to 0.2 and 0.1 in the down- and uplink, respectively. As indicated in Fig. 1 (bottom), we assume the TCP and HTTP proxy to be located somewhere within the core network, where the delay and IP loss probability from the proxy towards the UTRAN is zero.

3.2 Emulation Environment

The previously described scenario is mapped to the emulation setup shown in Fig. 2, which consists of four standard Linux-PCs. The heart of the emulation setup is the UTRAN emulation, which is based on the emulation library developed at the IKR [14]. IP packets are sent from the server-PC, which runs an Apache 1.3.29 web server, to the emulator. The emulator delays (and possibly drops) the IP packets according to the same Internet and core network model which is used for the simulation. Afterwards, the packets may be forwarded to the proxy-PC, which runs a Squid 2.5 HTTP proxy or a hand-written TCP proxy. Finally, the emulator delays the IP packets according to the UTRAN model and forwards them to the client-PC, which runs a Mozilla 1.6 web browser. Note that, in contrast to the model described in the previous section, there is a small delay due to the networking overhead between the proxy PC and the emulation PC. This is an acceptable circumstance, since the delay within the UTRAN is several orders of magnitude higher.

The Mozilla web browser was automated using an XUL-script, which automatically surfs a given list of web-pages. A new page was requested from the web server two seconds after a page and all its inline objects were completely loaded and displayed. The cache size of Mozilla was set to 1MByte, which is large enough to cache small images, such as arrows and bullet points, during one list cycle but too small to cache pictures belonging to any actual content.

To a large extend, Apache and Squid were run with its default configurations. Squid's *pipeline_prefetch* option, which supposedly improves the performance with pipelined requests, was turned on for all scenarios with activated HTTP pipelining. In a first trial, Squid was run in proxy-only mode, i.e. with deactivated

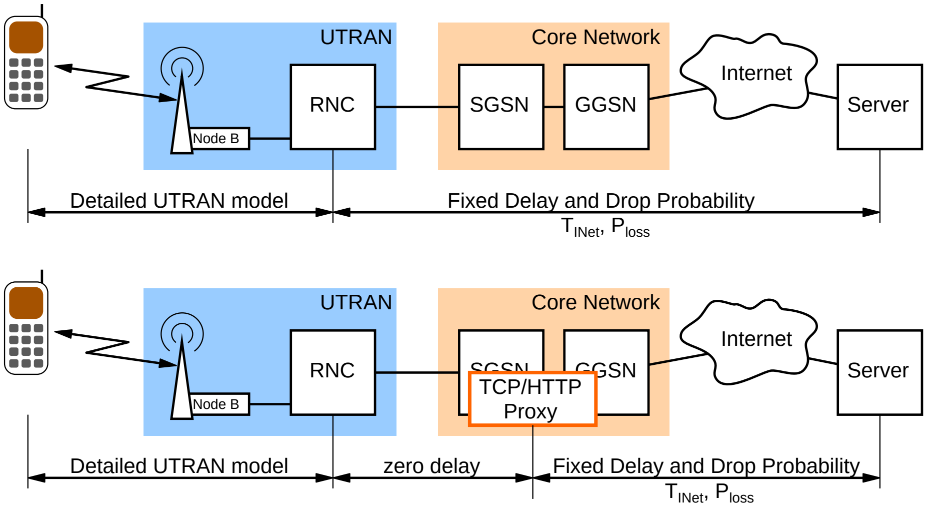


Figure 1. UMTS scenario without (top) and with TCP or HTTP proxy (bottom)

caching functionality. In a second trial, 1MByte of disc cache and 1MByte of memory cache were added, which again is large enough to cache small objects, but too small to cache actual content during one list cycle. Additionally, the Squid cache was forced to always use the objects from its cache, regardless of the expiry date reported from the web server. This implies that objects are only removed from the cache due to its limited storage capacity. We believe that this configuration quite well imitates the situation of a highly loaded proxy which has to serve millions of pages per day to many different users.

In contrast, the hand-written TCP proxy simply opens another TCP connection towards the server for each incoming connection from the client, and directly writes any data received from either side to the respective other socket.

Throughout our paper, we do not consider DNS lookups. On the one hand, DNS lookups can impose a significant waiting time at the beginning of a page load, especially if the network's Round Trip Time (RTT) is large (as it is in mobile networks). On the other hand, they have to be done only once upon the first access to a particular server within a session. Therefore, their frequency is hard to determine and model. We will therefore omit the effect of DNS lookups. As DNS lookups can be done by application layer proxies very efficiently, we have to keep in mind that this simplification favors scenarios with no proxy and with a TCP proxy.

3.3 FTP service scenario

In this paper, we investigate FTP traffic by means of simulation, where we only consider downlink bulk data transfer. The TCP flavor was *NewReno* with activated window scaling. For TCP bulk data transfer, it is sufficient to consider the TCP proxy case, since an application layer proxy (i.e. bulk data transfer over HTTP or FTP) would essentially yield the same results.

3.4 WWW service scenario

Since WWW traffic is very difficult to model for a simulative study, we chose to investigate it by means of emulation. For that purpose, we created a snapshot

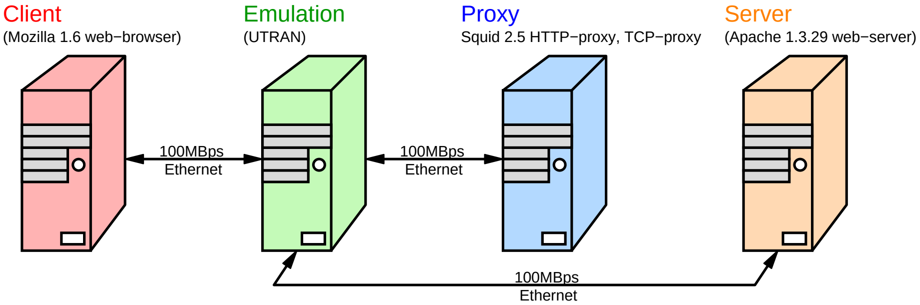

Client
(Mozilla 1.6 web−browser)
Emulation
(UTRAN)
Proxy
Squid 2.5 HTTP−proxy, TCP−proxy
Server
(Apache 1.3.29 web−server)
100MBps
Ethernet
100MBps
Ethernet
100MBps
Ethernet

Figure 2. Emulation Setup

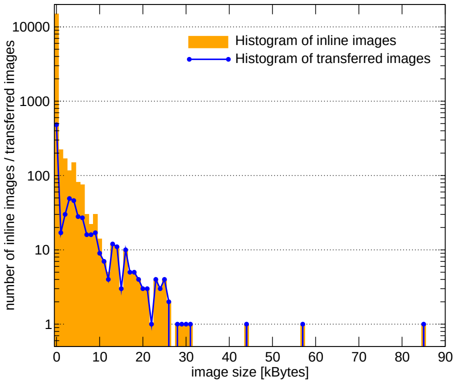


Figure 3. Histogram of inline images and transferred images

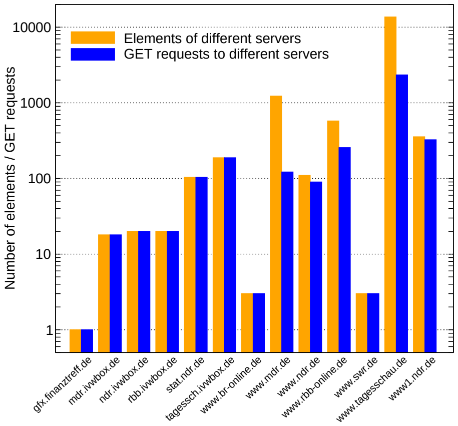


Figure 4. Histogram of elements of different servers and GET requests to different servers

of the web-site `www.tagesschau.de` and related sites on the local web server. Figure 3 shows the histogram of all inline images contained on all web-pages within one Mozilla list cycle. This histogram counts identical images on one particular web-page multiple times. Additionally, Fig. 3 contains the histogram of the number of actually transferred images, which are less due to the browser's internal cache. As it is the case with most web-sites, there is a strong tendency towards many small helper images, such as transparent pixels, bullet images and the like. Figure 4 shows the distribution of objects across the different involved domains. Additionally, the figure shows the distribution of the GET requests issued to the different domains within one list cycle. The majority of accesses goes to the main `tagesschau.de`-server. In addition, several web-pages and inline objects are fetched from related regional web-sites, such as `ndr.de` for news from northern Germany or `swr.de` for news from southern Germany. Moreover, there are several accesses to web-servers responsible for tracking access statistics (`ivwbox.de`, `stat.ndr.de`).

4 FTP Results

In this section, we evaluate the performance of a downlink bulk data transfer in dependence of the loss probability P_{loss} and the delay T_{INet} in the fixed network part. Figure 5 shows the total TCP throughput plotted over the delay T_{INet} for different loss probabilities P_{loss} and the two cases with and without proxy. Without a TCP proxy, the throughput dramatically decreases if the fixed network exhibits packet losses. Even for as few as 0.5% of lost packets in the fixed network, the throughput is reduced by more than 10%. If a TCP proxy is used, the situation is completely different. Now, the system can recover from packet losses in the fixed network part much quicker. Consequently, the TCP throughput remains at a high level for low to medium loss probabilities P_{loss}. Only for

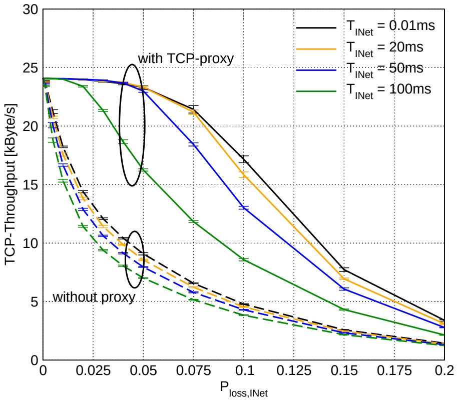


Figure 5. Performance of downlink bulk TCP traffic with and without TCP proxy

high loss probabilities or long delays T_{INet} does the system throughput decrease again. This goes well along with the results presented in [5].

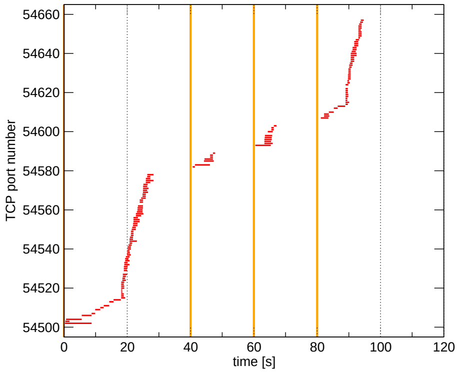
5 WWW Results

5.1 TCP connection behavior

We will first discuss Mozilla's TCP connection behavior for different system configurations. Figure 6 illustrates the TCP connection behavior for a direct connection to the web server with no proxy. The figure shows the duration of all TCP connections on their corresponding port numbers over the time[4]. No automated surfing was used here. Instead, the main page `www.tagesschau.de` was manually accessed at time t=0s, and subsequent subpages were accessed at times t=40s, t=60s and t=80s. The browser was terminated at t=100s. HTTP version was 1.1 with deactivated keep-alive and no pipelining. The browser's cache was cleared at the beginning of the measurement. This protocol configuration forces the web browser to open a separate TCP connection for each inline object which results in 414 TCP connections for loading all four pages. This implicates a considerable overhead with respect to delay and transmitted data, since each TCP connection needs one RTT to be established and at least one RTT to be torn down [15]. Especially in networks which exhibit a high RTT, the performance will suffer from this behavior.

The situation can be improved by activating the HTTP keep-alive feature [16] (persistent connections). Figure 7 shows the new pattern of TCP connections, with the destination server annotated for each TCP connection. The number of TCP connections reduces to only 16 in total. Note that TCP connections to the server hosting the main web-page are kept persistent for a long time, while

[4] Note that not all consecutive port numbers are used within the session.



time	accessed page
0s	`www.tagesschau.de`
40s	`www.tagesschau.de/regional`
60s	`www.tagesschau.de/hamburg/0,1209,SPM13150_NAVSPM11178,00.htm`
80s	`www1.ndr.de/ndr_pages_std/0,2570,OID460712_REF960,00.html`
100s	Mozilla was terminated.

Figure 6. Timeline of TCP connections with no proxy and no HTTP keep-alive

connections to servers serving inline objects are torn down much faster. However, this relatively fast connection teardown is not caused by the web browser, but by the web-server, i.e. the browser has no influence on it. Instead, it highly depends on the settings of the Apache web server, which provides two main parameters to control the teardown of persistent connections, namely *MaxKeepAliveRequests* and *KeepAliveTimeout*. The first parameter specifies the maximum number of objects that may be transmitted over a persistent connection before it is being closed, while the second parameter indicates the time after which a persistent connection is closed if it became idle. The default values of these parameters are 100 and 15 seconds, respectively. Since, most likely, there are many web-sites using the default values, we will use the same values for our studies. Figure 7 nicely reflects the value of *KeepAliveTimeout* for servers serving inline objects.

With HTTP keep-alive, the number of connections reduces, leading to less overhead with the potential of faster loading times. On the other hand, the client has to wait for the response from the server for each requested inline object before it can issue another request. This implies that there may be far fewer outstanding requests as compared to non-persistent connections, which is

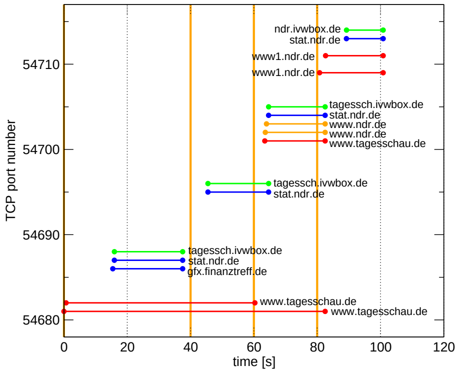


Figure 7. Timeline of TCP connections with no proxy and HTTP keep-alive

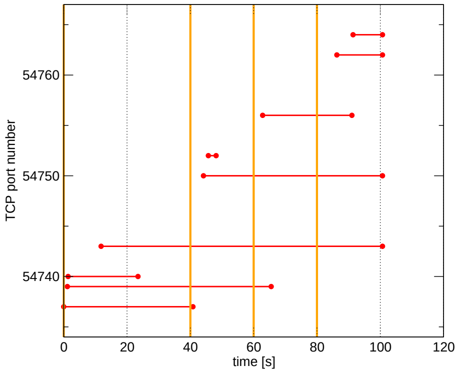


Figure 8. Timeline of TCP connections with HTTP proxy and HTTP keep-alive

a potential performance inhibitor in networks with large RTTs. This situation can be greatly improved by activating pipelining in HTTP 1.1, which allows the request of multiple inline objects over the same persistent connection without waiting for each response.

Finally, we will consider the connection behavior if an HTTP proxy is used. RFC 2616 [16] states that "talking to proxies is the most important use of persistent connections". The reason is that multiple different servers can be accessed via the same persistent client–proxy connection. The connection behavior in such a scenario is shown in Fig. 8. It is now no longer possible to associate a TCP connection with a particular server. The chart shows that the number of connections almost halfened to 9. Again, all but one of the connections that are closed before the end of the session are terminated by the WWW proxy.

We do not need to consider the case of a TCP proxy here, since it does not change the semantics of an HTTP connection.

5.2 Page loading times

In this section, we investigate the page loading times for ideal Internet conditions. In particular, we chose $T_{\mathrm{INet}} = 20\mathrm{ms}$ and $P_{\mathrm{loss}} = 0$. Table 1, Fig. 9 and Fig. 10 show the mean and median of the page loading times for all considered proxy scenarios and different HTTP 1.1 configurations. For the Squid scenario, we have provided two values per table cell. The first one was obtained with Squid acting as a proxy only, the second one with Squid additionally caching web objects. Since the difference is very small, Fig. 9 and Fig. 10 show only the values with caching enabled.

For the non-proxied scenario and the scenario with a TCP proxy, it is obvious that HTTP performs best when keep-alive and pipelining is activated, and performs worst if both is deactivated. In contrast, according to information in [17], Squid does not support pipelined requests towards servers very well. While it accepts pipelined requests from the client side, it transforms them into parallel requests, where no more than two requests from a pipeline can be fetched in parallel. This drawback is reflected in Table 1, since, with pipelining, performance does not improve compared to the non-pipelined case.

	no keep-alive		keep-alive		
	Mean	Median	Mean	Median	
no pipelining	8.2s	6.3s	7.73s	5.95s	no proxy
pipelining	—	—	6.22s	4.96s	
no pipelining	8.56s	6.47s	8.22s	6.16s	TCP proxy
pipelining	—	—	6.74s	4.94s	
no pipelining	8.85s / 8.81s	6.67s / 6.58s	7.02s / 6.92s	5.35s / 5.26s	Squid proxy
pipelining	—	—	7.09s / 6.91	5.41s / 5.40s	

Table 1. Mean values and median of the loading time

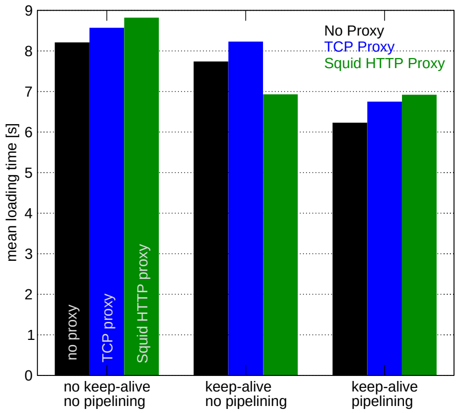


Figure 9. Mean page loading time

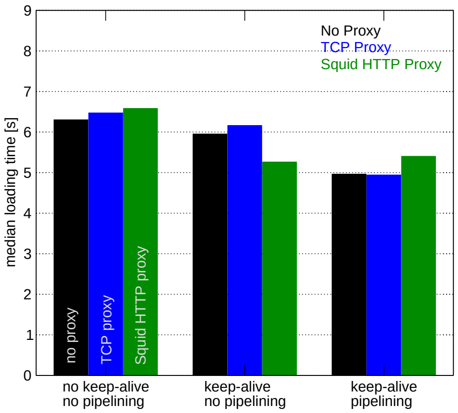


Figure 10. Median of page loading time

When comparing the different proxy scenarios, we can see that a proxy not necessarily improves the performance. For all considered HTTP configurations, the TCP proxy worsens the performance compared to the non-proxied case, which results from the overhead introduced by the proxy. However, the proxy could significantly increase the performance under two circumstances: First, if the fixed part of the connection was non-ideal, the TCP proxy could efficiently recover from lost packets within the fixed part of the network and also mitigate the effects of long RTTs in the fixed network (see section 5.3 and 5.4). The second circumstance is a proxy with an optimized TCP sender towards the UE. The most efficient and simple measure hereby certainly is an increase of the initial congestion window, since a small congestion window will prevent the full usage of the available bandwidth within the RAN for a relatively long time at the beginning of a TCP connection. This results from the long RTT in mobile networks (see [18] for measurements in GPRS networks on this issue).

The same can be said for the Squid scenario and non-persistent HTTP connections with no pipelining. If keep-alive is activated, we observe a significant performance increase compared to the non-proxied scenario. This goes well along with the observations made in section 5.1: The UE can now maintain persistent HTTP connections to the proxy, even if the requests are issued across different servers. This leads to fewer TCP connection establishments and teardowns across the RAN. If pipelining is activated, the performance in the non-proxied scenario significantly increases, whereas the performance with Squid remains about the same. The reason is again the lack of pipelining support in Squid.

Figures 11 and 12 show the complementary cumulative distribution function (CCDF) of the page loading time for the different proxy scenarions with deactivated and activated keep-alive, respectively. Pipelining was deactivated in both cases. In both diagrams, we can identify three groups of pages. About 40% of all pages are loaded within 5 seconds. 50% are loaded faster than 12 – 15 seconds, depending on the scenario. Only 10% or less of all pages take longer than 15 seconds to load. For the case of non-persistent connections, all CCDFs are very

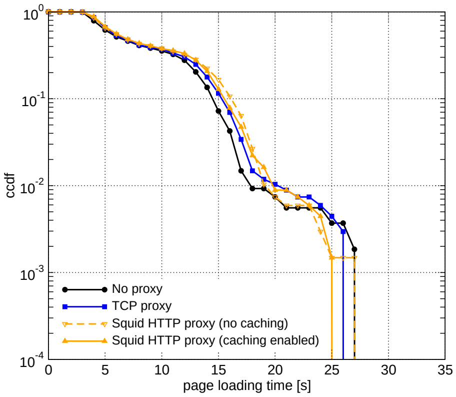


Figure 11. CCDF of page loading times, no HTTP keep-alive and no pipelining

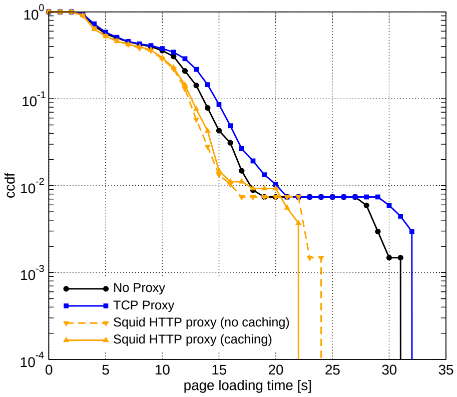


Figure 12. CCDF of page loading times, HTTP keep-alive without pipelining

close together, whereas the performance gain with an HTTP proxy is well visible if persistent connections are allowed. In contrast, for the case of persistent connections, there is a noticable difference between the performance of the HTTP proxy and the other two scenarios. This again reflects the great advantage that HTTP proxies can take of persistent connections.

We already know that the mean and median of the page loading time becomes smaller if persistent connections are activated. However, when comparing Fig. 11 and 12, we can observe a much heavier tail in the CCDFs for persistent connections as compared to the non-persistent case. In other words, there are a few pages that have a significantly longer loading time if persistent connections are allowed. In fact, the heavy tail is caused by only one page, which is the main page `www.tagesschau.de`. This page is characterized by a number of inline objects well above average. It is a good example of where the more serialized issuing of HTTP requests with persistent connections reduces performance.

5.3 Internet packet losses

Studies of the Internet packet dynamics, such as in [19], and recent measurements [20] reveal that typical IP-packet loss probability on the Internet are on the order of 0 to 5 percent, or even more in very bad cases.

Figures 13 and 14 plot the mean and median of the page loading time over the Internet packet loss P_{loss} if keep-alive is activated but pipelining is deactivated. Figure 15 and 16 shows the same but with activated pipelining. Squid is considered with caching enabled only. As it can be expected from previous studies (e.g. [5]), the mean and median of the page loading time increase in the non-proxy case as P_{loss} increases. It is interesting to observe that this increase is approximately linear with P_{loss}.

A similar increase can be observed if a TCP proxy is used. However, the increase is much smaller. We already noted that the performance with TCP

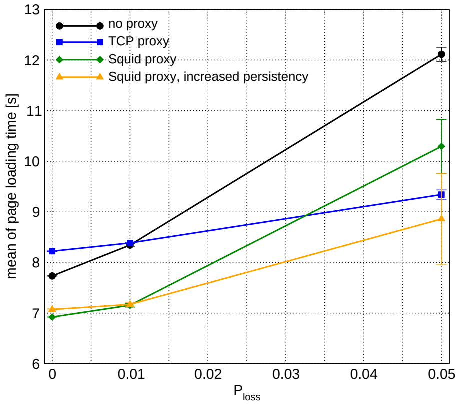


Figure 13. Mean of page loading times, HTTP keep-alive without pipelining

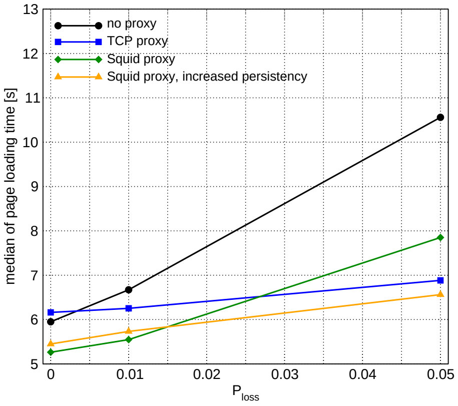


Figure 14. Median of page loading times, HTTP keep-alive without pipelining

proxy is worse compared to the non-proxied case if $P_{\text{loss}} = 0$. As P_{loss} increases, the advantage of the proxy becomes obvious, as it outperforms the non-proxied case for $P_{\text{loss}} > 0.01$. We should expect a similar behavior for an HTTP proxy. However, we observe a strong performance decrease in the Squid scenario as P_{loss} increases. For both HTTP configurations, the Squid performance eventually drops far below the performance of the TCP proxy. This behavior is not intuitive, since both proxies split the TCP connection and should be able to quickly recover from packet losses in the Internet.

The reason for this is the following. In the non-proxied case, for each displayed page, the web browser has to issue many GET requests to the web server. That means, persistent HTTP connections are usually busy. Hence, they will not be closed by the web server due to a timeout caused by the *KeepAliveTimeout* timer, but only due to reaching the maximum number of transmitted inline objects on a

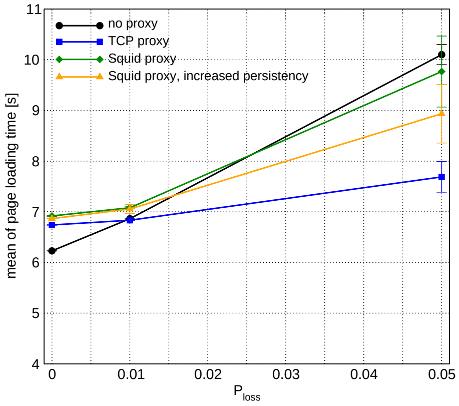


Figure 15. Mean of page loading times, HTTP keep-alive with pipelining

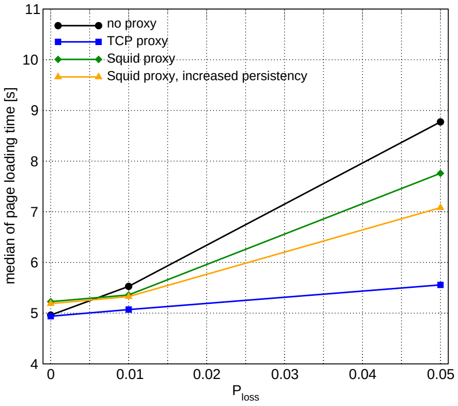


Figure 16. Median of page loading times, HTTP keep-alive with pipelining

connection. Consequently, the TCP connections will be in congestion avoidance most of the time, unless multiple TCP packets are lost per RTT. The same thing applies to the TCP proxy scenario. In contrast to this, the Squid proxy has an essentially different connection behavior towards the web server. Since it caches most of the web objects, the persistent connections towards the web server are mostly idle and frequently get closed by the web server. This implies that the connections towards the web server are in slow-start very often. That is, a TCP connection is more likely to be in slow-start when a packet loss occurs, which results in long recovery times[5]. The final consequence is that the link from the HTTP proxy to the web server may be idle for quite a long time, while the web browser is still waiting for objects to be served from the proxy. This eventually leads to the bad performance in the Squid scenario if packet losses in the Internet are present.

Squid performance can be improved by setting the Apache parameters *MaxKeepAliveRequests* and *KeepAliveTimeout* to higher values. Here, we set it to infinity and 150s, respectively. Now, Apache waits much longer before closing any persistent connections. Figures 13 through 16 contain the measured results obtained in the Squid scenario, labeled with *increased persistency*. Looking at the non-pipelined case, the Squid proxy now behaves as we expect it: the mean loading times are below or about equal to those of the TCP proxy, which is intuitive, since the HTTP proxy can cache content. The pipelined case shows some improvements, but the performance cannot match that of the TCP proxy due to the reasons discussed in section 5.2.

5.4 Internet packet delay

Finally, we will investigate the influence of the packet delay T_{INet} in the fixed network part. Figures 17 through 19 plot the mean page loading time over the loss probability P_{loss} for different delays T_{INet} and the three considered proxy scenarios. It is obvious, that the mean page loading time must increase as T_{INet} increases. It is more interesting to note that, for each scenario, the slope of the curves is independent of the delay T_{INet}. That is, the delay T_{INet} has an additive impact on the loading times. This indicates that the persistent TCP connections are mostly in congestion avoidance.

As expected, the advantage of the proxy solutions becomes greater as the delay T_{INet} increases. The reason is again the faster recovery from packet losses in the proxy case, especially if the delay is large. We can also see that an HTTP proxy can take more advantage of its caching functionality if the delay is large, as it takes longer to retrieve documents and objects from the actual web server.

6 Conclusion

We investigated the connection behavior of the Mozilla web browser and evaluated the performance of TCP and HTTP proxies under typical web traffic in

[5] We measured TCP timeout durations of 10s and more.

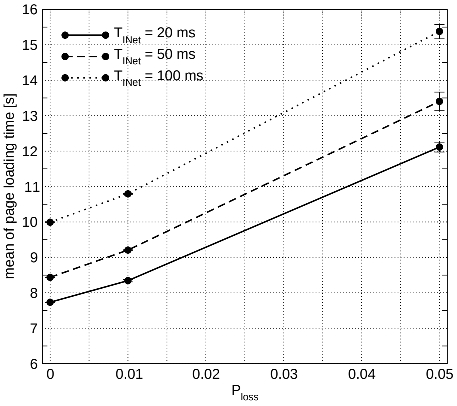


Figure 17. Mean of page loading times, no proxy, HTTP keep-alive without pipelining

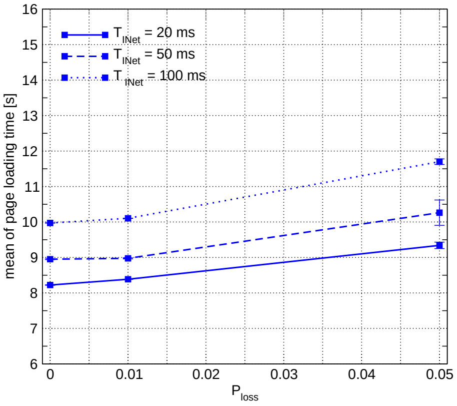


Figure 18. Mean of page loading times, TCP proxy, HTTP keep-alive without pipelining

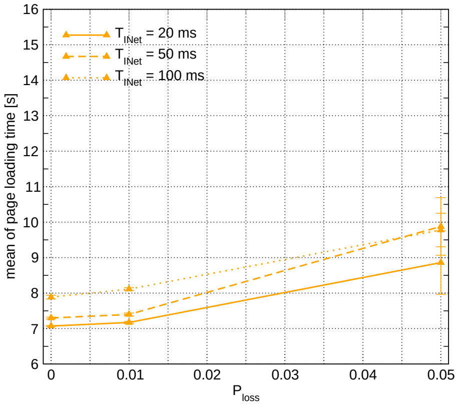


Figure 19. Mean of page loading times, Squid HTTP proxy with increased persistency, HTTP keep-alive without pipelining

an UMTS environment. Our studies show that proxies do not increase system performance by default. Instead, the proxy concept and the parameters of all involved devices must be carefully chosen under consideration of all network aspects. In the case of a good server connection, HTTP performs best with activated keep-alive and pipelining and without any proxy. If the connection towards the server is bad, the same HTTP configuration in combination with a TCP proxy delivers best results, whereas the Squid HTTP proxy does not perform well due to the lack of pipelining support.

Acknowledgements

Michael Scharf is funded by the German Research Foundation (DFG) through the Center of Excellence (SFB) 627. The authors would like to thank Jochen Zwick for his valuable work on the automation of the Mozilla web browser.

References

1. J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby: "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, June 2001.
2. A. Bakre and B. R. Badrinath: "I-TCP: Indirect TCP for Mobile Hosts", in Proc. *Proc. 15th Int'l. Conf. on Distr. Comp. Sys.*, May 1995.
3. R. Yavatkar and N. Bhagawat: "Improving End-to-End-Performance of TCP over Mobile Internetworks", in *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, Dec. 1994.
4. R. Chakravorty and I. Pratt: "Rajiv Chakravorty and Ian Pratt, WWW Performance over GPRS", in *Proc. IEEE MWCN 2002*, Sep. 2002, Stockholm, Sweden.
5. M. Meyer, J. Sachs and M. Holzke: "Performance Evaluation of a TCP Proxy in WCDMA Networks", *IEEE Wireless Communications*, Vol. 10, Iss. 5, pp. 70–79, Oct. 2003
6. H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov and F. Khafizov: "TCP over second (2.5G) and third (3G) generation wireless networks", RFC 3481, Feb. 2003.
7. A. Mutter, M. C. Necker and S. Lück: "IP-Packet Service Time Distributions in UMTS Radio Access Networks", in *Proc. EUNICE 2004*, Tampere, Finland.
8. M. Scharf, M. C. Necker and B. Gloss: "The Sensitivity of TCP to Sudden Delay Variations in Mobile Networks", LNCS 3042, pp. 76 – 87, May 2004.
9. H. Balakrishnan, S. Seshan und R. H. Katz: "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks", *Wireless Networks* 1(4), pp. 469 – 481, Feb. 1995.
10. J. Schüler, S. Gruhl, T. Schwabe and M. Schwiegel: "Performance Improvements for TCP in Mobile Networks with high Packet Delay Variations", in Proc. *Int. Teletraffic Congress (ITC-17)*, Sept. 2001.
11. M. C. Chan and R. Ramjee: "TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation", in Proc. *ACM MobiCom 2002*, Sept. 2002.
12. K. Brown and S. Singh: "M-TCP: TCP for Mobile Cellular Networks": *ACM SIGCOMM Computer Communications Review*, 27(5), pp. 19 – 43, Oct. 1997.
13. http://www.ikr.uni-stuttgart.de/IKRSimLib/
14. S. Reiser: "Development and Implementation of an Emulation Library", Project Report, University of Stuttgart, IKR, September 2004.
15. W. R. Stevens: "TCP/IP Illustrated, Volume 1", Addison-Wesley, 1994.
16. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee: "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616.
17. `http://www.squid-cache.org`, September 2004.
18. R. Chakravorty, J. Cartwright and I. Pratt: "Practical Experience With TCP over GPRS", in *Proc. IEEE GLOBECOM*, Nov. 2002.
19. V. Paxson: "End-to-End Internet Packet Dynamics", *Trans. on Networking*, Vol. 7, No. 3, pp. 277–292, June 1999.
20. `http://www.internettrafficreport.com`, September 2004.