Impact of Multi-path Routing on TCP Performance

Jonas Karlsson, Per Hurtig, Anna Brunstrom and Andreas Kassler
Dept. of Computer Science
Karlstad University
Karlstad, Sweden
Email: {jonas.karlsson, per.hurtig, anna.brunstrom, andreas.kassler}@kau.se

Giovanni Di Stasi
Dept. of Computer Science
University "Federico II" of Naples
Naples, Italy
Email: giovanni.distasi@unina.it

***Abstract*—Routing packets over multiple disjoint paths towards a destination can increase network utilization by load-balancing the traffic over the network. The drawback of load-balancing is that different paths might have different delay properties, causing packets to be reordered. This can reduce TCP performance significantly, as reordering is interpreted as a sign of congestion. Packet reordering can be avoided by letting the network layer route strictly on flow-level. This will, however, also limit the ability to achieve optimal network throughput. There are also several proposals that try to mitigate the effects of reordering at the transport layer. In this paper, we perform an initial evaluation of such TCP reordering mitigations in multi-radio multi-channel wireless mesh networks when using multi-path routing. We evaluate two TCP reordering mitigation techniques implemented in the Linux kernel. The transport layer mitigations are compared using different multi-path routing strategies. Our findings show that, in general, flow-level routing gives the best TCP performance and that transport layer reordering mitigations only marginally can improve performance.**

***Keywords*-TCP; reordering; wireless mesh networks; multi-channel; multi-radio; multi-path**

I. Introduction

Wireless mesh networks (WMNs) are considered to be a new and promising technique to provide Internet connectivity for cities, rural areas or user-communities. To improve both capacity and reliability in such networks each node can be equipped with multiple radio interfaces using a set of orthogonal channels. This channel diversity improves the performance of a single-path, while at the same time increases the possibility to create multiple interference-free paths between the sender and the receiver. In this paper we focus on the use of multiple paths to achieve load-balancing and thus increase application performance.

Multi-path routing algorithms can be used for load-balancing by utilizing disjoint links and paths. This path allocation can be done on a per-flow or per-packet basis. Both approaches have advantages and limitations. For instance, per-flow level allocation might be unable to fully utilize the network as it can be difficult to evenly load-balance the traffic amongst all paths. However, when using TCP the data packets and the acknowledgments can be treated as two different flows. Therefore, per-flow level allocation can be used for the data packets and the acknowledgments can be allocated on a per-packet basis, and vice versa. A packet-based allocation scheme, on the other hand, allows a more fine-grained control over the resources compared to per-flow allocation. However, load-balancing packets from the same flow over several paths, with different delay properties, might cause packets to arrive reordered. For TCP, packet reordering is a well-known problem that can seriously hamper application performance [1]. To solve such performance problems, several techniques have been proposed to mitigate the effects of reordering, e.g. [2].

To find the performance trade-off between packet reordering and network utilization, we approach the problem from two directions. First, we evaluate both flow-based and packet-based path allocation. Second, we use three different TCP implementations, including a TCP variant without mechanisms to mitigate reordering effects, the standard Linux TCP implementation with built-in reordering mitigation techniques and TCP with Non-Congestion Robustness (NCR) [2] extensions. We consider a general multi-path routing protocol that requires only minor knowledge about the upper layers, i.e. flow information.

Our findings show that when the amount of TCP-flows increases, flow-based path allocation becomes superior to using packet-based allocation, regardless if TCP is equipped with mechanisms to mitigate the effects of reordering or not.

The rest of the paper is structured as follows. Section II discusses packet reordering and possible mitigation techniques. Section III evaluates these techniques and Section IV concludes the paper.

II. TCP and Packet Reordering

Packet reordering is a rather common event that poses negative effects on applications and protocols that require in-order data delivery. For TCP, the reordering of both data and acknowledgments affects performance [3]. Previous research has shown the possibility to mitigate reordering effects in a reactive and/or proactive manner. Reactive algorithms [4] try to undo state changes caused by reordering, e.g. congestion control invocations due to unnecessary fast retransmits, while pro-active mechanisms [2] try to inhibit such state changes in the first place. Some algorithms use both approaches to provide even better protection [5]. While

978-1-4673-1239-4/12/$31.00

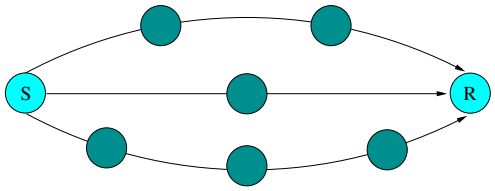


Figure 1. WMN Topology.

most of the proposed mitigation techniques are not widely implemented, the built-in techniques of the Linux kernel and the TCP-NCR algorithms [2] can be seen as the most accepted solutions.

The Linux TCP implementation offers protection against reordering by using a mixed mitigation approach. The reactive part tries to undo congestion control decisions that were made unnecessarily. The pro-active part tries to inhibit unnecessary fast retransmissions by dynamically increasing the duplicate acknowledgment threshold. This increase is based on the maximum observed reordering length so far, and the threshold can be as large as 127. TCP-NCR is a pro-active scheme that extends the loss detection phase to better determine if out-of-order deliveries are due to packet loss or packet reordering. The detection phase is extended to roughly one round-trip time, which is accomplished by setting the duplicate acknowledgment threshold to approximately one congestion window.

III. TCP Performance in a Simple WMN Topology

In this section, we evaluate the standard Linux TCP implementation (Linux) and the TCP-NCR algorithm (NCR), and compare their performance against a version of the Linux TCP implementation that does not contain protection against reordering (NewReno). All evaluated TCP implementations use the NewReno congestion control, as there are incompatibilities between TCP-NCR and CUBIC [6] in the 2.6.26 version of the Linux kernel.

A. Simulation Setup

Figure 1 illustrates the topology that was used in the experiment. As the paths between the sender and the receiver are of different lengths, the total delay of a packet transmission depends on which path is used for forwarding. Thus, excluding flow-based allocation, reordering is going to be present.

We used four allocation strategies: flow-based allocation (data+ack), where both data and acknowledgments of a flow were fixed to single paths; flow-based data allocation (data), where all TCP data packets of a flow were fixed to a single path and acknowledgment packets were allocated dynamically to different paths; flow-based acknowledgment allocation (ack), where only the TCP acknowledgments of a flow were fixed to a single path and data packets were allocated dynamically to different paths; and packet-based allocation (packet), where both data and ack packets where allocated dynamically. In the experiment the desired path utilization was equal, which reduces the L2.5R forwarding scheme used [7] to a round-robin scheme. For flow-level allocation the path is determined by the first packet of a new flow.

To conduct the experiments we used ns version 2.32 together with the network simulation cradle (NSC) version 0.5.2 [8], which enables the use of the real Linux TCP implementation. The MAC/PHY layer was the default ns-2 IEEE 802.11 MAC/PHY layer configured to simulate an IEEE 802.11a network card using a PHY layer speed of 54 Mbit/s. All nodes were equipped with multiple radios, configured to orthogonal channels. We varied the number of TCP flows between 1 and 6, where each flow lasted for about 300 seconds. Each experiment was repeated 30 times. All graphs contain 95% confidence intervals, but these are too small to be visible.

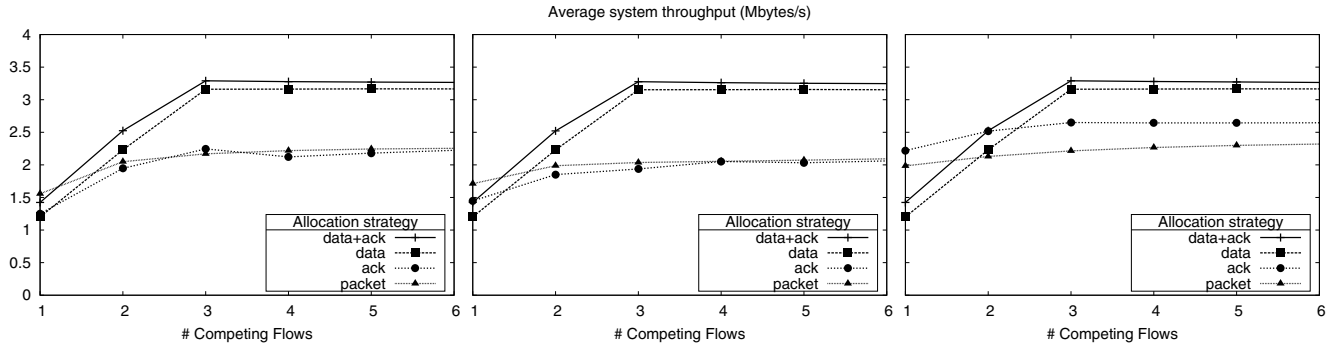
B. Results

Figure 2 shows the average system throughput (in Mbytes/s) as a function of the number of competing flows for the different allocation strategies and TCP versions. There are two distinct sections in all the graphs. When there are less than three flows, (data+ack) cannot fully utilize all three paths. When there are three or more flows, it is possible to use (data+ack) and still utilize the underlying capacity fully. However, this is not possible for (packet) as the maximum sustainable throughput is limited by the effects of reordering.

Figure 2(a) shows the results for NewReno, which performs almost identical to Linux, shown in Figure 2(c), when (data+ack) and (data) is used. However, for (packet) Linux performs slightly better than NewReno. Furthermore, when (ack) is used, Linux achieves noticeably higher throughput than NewReno. In both cases the reason for the performance advantage of Linux is the built-in mitigation techniques of Linux that are able to sustain better performance during reordering.

The most interesting result is that for Linux (ack) performs much better than (packet). The reason for this is that the reordering mitigation scheme in Linux heavily relies on receiving acknowledgments correctly. If an acknowledgment is reordered and arrives too late, it may be discarded by TCP. In such a situation, the possible reordering information contained in the acknowledgment cannot be used by Linux.

Regardless, in absolute numbers Linux does only perform slightly better than NewReno. There are several reasons for this. First, as the duplicate acknowledgment threshold is increased in response to detected reordering, it will eventually become too large for lost packets to be fast retransmitted. In such situation, Linux is forced to wait for a lengthy retransmission timeout (RTO). When the RTO finally occurs, the threshold is reset to three, which will make Linux vulnerable to unnecessary retransmissions



(a) NewReno (b) NCR (c) Linux

Figure 2. Average System Throughput for Different Reordering Mitigation Techniques and allocation strategies.

until the threshold has converged again. Second, when the duplicate acknowledgment threshold is increased, Linux will actually allow more reordering to happen, as compared to NewReno, which hurts the performance. This is because the reordering mitigations will help TCP to sustain a large congestion window. Thus, the important observation here is that when packets are allowed to be reordered there will be more packets outstanding that can be reordered.

The performance of NCR is shown in Figure 2(b). As indicated by the graphs, the performance of NCR is similar to that of NewReno. The reason for NCR's poor performance is related to the design of the scheme. NCR is unable to grow the congestion window sufficiently, if compared to Linux. There are two reasons for this. The first is related to how NCR behaves when it enters the extended loss detection phase; it exits slow-start whenever a duplicate acknowledgment arrives. This causes NCR to require much longer time to probe the available capacity of the underlying network, given that reordering occurs in the beginning of the transmission. NCR is also unable to increase the congestion window when receiving acknowledgments that mark the end of one reordering event, and at the same time the beginning of a new one. Therefore, when reordering occurs frequently, NCR might force TCP to have a small congestion window during the whole transfer.

IV. Conclusions

In this paper, we have evaluated TCP performance using different multi-path traffic allocation strategies and TCP reordering mitigation techniques. This allows the benefit of load-balancing the traffic over multiple paths while avoiding the drawback of packet reordering. The Linux TCP implementation, with built-in reordering robustness, was able to benefit from allocation decisions on a per-packet basis in a simple scenario with a single flow. In all other tested settings, TCP throughput was equal or higher with flow-based allocation. We are currently evaluating more complex topologies and traffic patterns. Furthermore, as our current evaluation is limited to single path TCP, our future work will also evaluate multi-path transport layer protocols, e.g. MPTCP [9].

Acknowledgment

This research is supported by grant YR2009-7003 from Stiftelsen för internationalisering av högre utbildning och forskning (STINT).

References

[1] K.-C. Leung, V. O. K. Li, and D. Yang, "An overview of packet reordering in transmission control protocol (TCP): Problems, solutions, and challenges," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, April 2007.

[2] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton, "Improving the robustness of TCP to non-congestion events," *Internet RFCs, ISSN 2070-1721, RFC 4653*, August 2006.

[3] J. C. R. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, December 1999.

[4] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, January 2002.

[5] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A reordering-robust TCP with DSACK," in *Proceedings of the 11th IEEE International Conference on Networking Protocols (ICNP)*, Atlanta, USA, November 2003.

[6] S. Ha, I. Ree, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, July 2008.

[7] S. Avallone, I. Akyildiz, and G. Ventre, "A channel and rate assignment algorithm and a layer-2.5 forwarding paradigm for multi-radio wireless mesh networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, February 2009.

[8] S. Jansen and A. McGregor, "Simulation with real world network stacks," in *Proceedings of the 37th Winter Simulation Conference (WSC)*, Orlando, USA, December 2005.

[9] A. Ford, C. Raiciu, M. Handley, S. Barré, and J. Iyengar, "Architectural guidelines for multipath TCP development," *Internet RFCs, ISSN 2070-1721, RFC 6182*, March 2011.