# The Interaction Between TCP Reordering Mechanisms and Multi-path Forwarding in Wireless Mesh Networks

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. In wireless mesh networks, multi-radio multi-channel nodes are often used to create a larger set of interference-free paths thus increasing the chance of load-balancing. 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 forward traffic strictly on flow-level. This would avoid the negative drawbacks of packet reordering, but will also limit the ability to achieve optimal network throughput. On the other hand, there are several proposals that try to mitigate the effects of reordering at the transport layer. In this paper, we perform an in-depth evaluation of such TCP reordering mitigations in multi-radio multi-channel wireless mesh networks when using multi-path forwarding. We evaluate two TCP reordering mitigation techniques implemented in the Linux kernel. The transport layer mitigations are compared using different multi-path forwarding strategies. Our findings show that, in general, flow-level forwarding 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 forwarding traffic over disjoint links and paths. This path allocation can be done on a per-flow or per-packet basis. Both approaches have advantages and limitations. When the number of flows is limited, per-flow allocation may not fully utilize the network. Furthermore, this form of allocation requires routers to track flow information and support mechanism to handle stale flows. 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 number 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. However, the TCP performance, using multi-path routing, highly depends on the network topology and traffic profiles, not so much on whether TCP is equipped with mechanisms to mitigate the effects of reordering.

The rest of the paper is structured as follows. Section II provides a background on multi-path routing in wireless networks. Section III discusses the negative effects that multi-path routing might have on TCP performance and how it is possible to mitigate these effects. This section also introduces the mitigation techniques that we consider in this paper. Section IV evaluates the considered TCP implementations, in a simple topology to determine how performance

978-1-4673-1430-5/12/$31.00 

is affected by different path allocation strategies. Section V then repeats the evaluation in a realistic WMN topology, using more complex traffic patterns. Section VI discusses related work. Finally, Section VII concludes the paper.

## II. Multi-path Routing

Standard routing protocols discover a single route between a source and a destination. In wireless multi-hop networks such as mesh networks, routes may break frequently due to fading or congestion. Multi-path routing is an interesting alternative as it allows discovering multiple paths for each source/destination pair. Such path diversity can be used to provide fault tolerance and higher aggregate bandwidth. Especially load balancing is an interesting option as it spreads traffic along multiple paths thus avoiding congestion and bottleneck links. Due to the shared nature of wireless communication, multi-path routing strategies may be difficult to design as interference from nearby nodes limits the performance.

For multi-path routing the following steps are required: route discovery, route maintenance and traffic forwarding. The route discovery step finds multiple routes between each source and destination pair. Typically, protocols want to find node disjoint or link disjoint paths to minimize interference and to provide higher fault tolerance. Even if node- or link-disjoint paths are found, transmissions on different links still interfere if the links are in the same collision domain. However, when multiple radios and channels can be used, such interference can be minimized through a proper channel assignment algorithm. Once traffic is sent along multiple routes, paths may break due to link or node failures. Then route maintenance needs to find alternative routes. There are different approaches to when such route maintenance should be triggered, due to the availability of different paths.

Traffic forwarding refers to the strategy of the node to select one or more out of the multiple next hops to send the packets towards the destination along the multiple paths. This is decided based on the allocation strategy. As an example, a flow-based allocating strategy would pin all the packets of one flow to a single path. A packet-based allocation strategy would distribute different packets from all flows amongst the paths found by the route discovery. Clearly, a packet-based allocation scheme allows a more fine-grained control over the resources compared to pinning flows to single paths. Mainly because it can be difficult to evenly load-balance the traffic amongst all paths when the offered rate of the flows may be different. However, when packet-based allocation is used and paths have different delay and capacity properties, packets might arrive out-of-order at the receiver, possibly leading to low transport layer performance when TCP is used. Note, that when using TCP, the data packets and acknowledgments can be treated as two different flows. So even a strategy where data packets are pinned to a single path while acknowledgments are allocated on a per packet basis is possible.

Especially for multi-radio/channel WMNs, the allocation of traffic is an important point to consider. Typically, the channel assignment determines the links within a collision domain that interfere and therefore the capacity. In order to avoid excessive collisions, it is important to not send more traffic over the links than supported by the channel assignment. An important scheme which achieves that is [3] (denoted here as L2.5R). The channel assignment is conducted by the FCRA (Flow-based Channel and Rate Assignment) algorithm [3] on a set of flow rates which represent the desired utilization of the link. The channels are assigned to links in order to make the precomputed flow-rates schedulable. Finally, in case flow rates cannot be served accordingly, they are scaled down to obtain a feasible solution. Once channels are assigned to nodes, the route discovery finds for each source/destination pair a set of paths centered around the shortest hop path. Only paths which fulfill a certain constraint in terms of number of hops are included in the multi-path. The packet scheduler in the nodes distributes traffic over the multiple paths in proportion to the adjusted flow-rate as calculated by the channel assignment algorithm. In order to do so, each node records the amount of traffic sent on each link to its neighbors, and chooses for each packet the neighbor among the candidate next-hops according to a minimum cost metric. Besides taking the bandwidth available on links into account, L2.5R has the potential for fast recovery from node/link failures. This is because it is not required to wait for nodes to re-compute the routing tables, as the node adjacent to the failed node/link can promptly blacklist the failed neighbor and balance the traffic among the remaining outgoing links. We use the L2.5R packet scheduler in the following as the base for the forwarding decisions but extend it in order to be able to pin certain flows (TCP data and/or acknowledgments) to a single next hop within the candidate next-hop set. More details on L2.5R can be found in [3].

## III. 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 affect performance [4]. Reordered data can be misinterpreted as packet loss by TCP's fast retransmit mechanism [5], causing unnecessary retransmissions and invocation of congestion control procedures. While retransmissions and reductions of the congestion window only occur if the reordering is severe enough to generate three duplicate acknowledgments at the receiver, shorter reordering events also affect TCP. For example, as TCP guarantees in-order data delivery all packets arriving out-of-order must be buffered, potentially requiring large receive buffers. Furthermore, when acknowledgments are reordered

the transmission becomes bursty and the congestion window may have difficulties in growing properly, resulting in poor performance.

To mitigate performance issues related to reordering, a number of approaches can be taken at the transport-layer. While we refer to such approaches as mitigation techniques/schemes, they typically try to inhibit the negative effects of reordering, not the reordering itself. Furthermore, most mitigation schemes are focused at solving performance problems related to unnecessary congestion control invocations, as this is regarded as the major performance problem related to reordering.

Typically, it is possible to classify a mitigation scheme as reactive, pro-active or mixed. Reactive schemes, such as [6], are often designed to revert TCP state changes that were made under the false assumption that packets were lost, when in fact reordering occurred. This type of mitigation technique can help TCP to maintain the sending rate by restoring the congestion window, when reordering is detected. Pro-active schemes, such as [2], [7], are instead designed to inhibit reordered packets from triggering the TCP loss recovery and thus the congestion control. To accomplish this, such proposals often extend the loss detection phase in TCP to allow more time to distinguish between loss and reordering. Practically, such loss detection extensions are often achieved by increasing the number of duplicate acknowledgments needed to trigger fast retransmit, i.e., the duplicate acknowledgment threshold. The increase can be based on previously observed reordering events, on the amount of outstanding data or other relevant metrics. As the reactive and pro-active approaches are fairly orthogonal, many mitigation schemes are designed to mix them to offer better overall robustness to reordering, e.g. [8], [9].

While most of the proposed mitigation techniques are not widely implemented, the built-in techniques of the Linux kernel [8] and the TCP-NCR algorithms [2] can be seen as the most accepted solutions. The former because of its widespread use, as being enabled by default in the Linux kernel, and the latter as it is formalized by the Internet Engineering Task Force (IETF) through RFC 4653.

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. To detect unnecessary invocations of the congestion control, Linux uses either Duplicate-SACKs (D-SACKs) [10] or the TCP timestamps (TS) extension [11]. When one of these mechanisms discovers that an unnecessary retransmission has been conducted, due to reordering, the congestion state is simply reverted. The pro-active part tries to inhibit unnecessary retransmissions by dynamically increasing the duplicate acknowledgment threshold. This is simply based on the maximum observed reordering length so far, and can be as large as 127. When a packet loss really does happen, and the fast retransmit mechanism is not able to detect it, the duplicate acknowledgment threshold is reset to three.

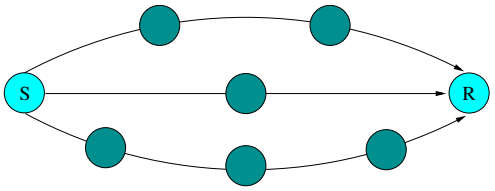



Figure 1. Simple WMN topology.

TCP-NCR is a pro-active scheme that extends the loss detection phase to better determine if duplicate acknowledgments 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. There are two versions of TCP-NCR: one aggressive version that maintains the original transmission rate during the detection phase and one careful version that halves the rate during the detection phase.

## IV. TCP Performance in a Simple WMN Topology

In this section, we evaluate and compare three different TCP versions in a simple WMN topology. The reason to use a simple synthetic topology is to isolate and understand the different mechanisms that influence TCP performance. First, as an experimental baseline we use a modified version of the Linux TCP implementation that does not contain protection against reordering (NewReno). Then, we evaluate the standard Linux TCP implementation (denoted as Linux) and the aggressive version of the TCP-NCR algorithm (denoted as NCR), and compare their performance to our baseline. All evaluated TCP implementations use the NewReno congestion control, as there are implementation incompatibilities between TCP-NCR and CUBIC [12] in the used 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

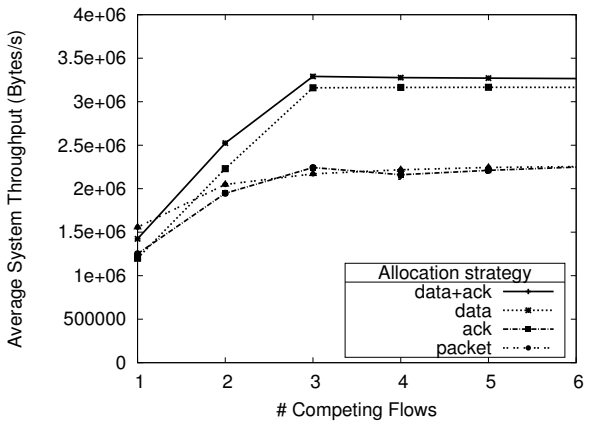



Figure 2. Average system throughput for NewReno.

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 [3] 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 [13] together with the network simulation cradle (NSC) version 0.5.2 [14], which enables the use of a 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 Bytes/s), for NewReno, as a function of the number of competing flows and the different allocation strategies. There are two distinct sections in Figure 2. When there are less then three flows, (data+ack), (ack) and (data) cannot fully utilize all three paths. When there are three or more flows, it is possible to conduct flow-based allocation and still utilize the underlying capacity fully. Consequently, (data+ack) and (data) are able to achieve near optimal utilization of the network. However, this is not possible for (packet) or (ack) allocation as the maximum sustainable throughput is limited by the effects of reordering.

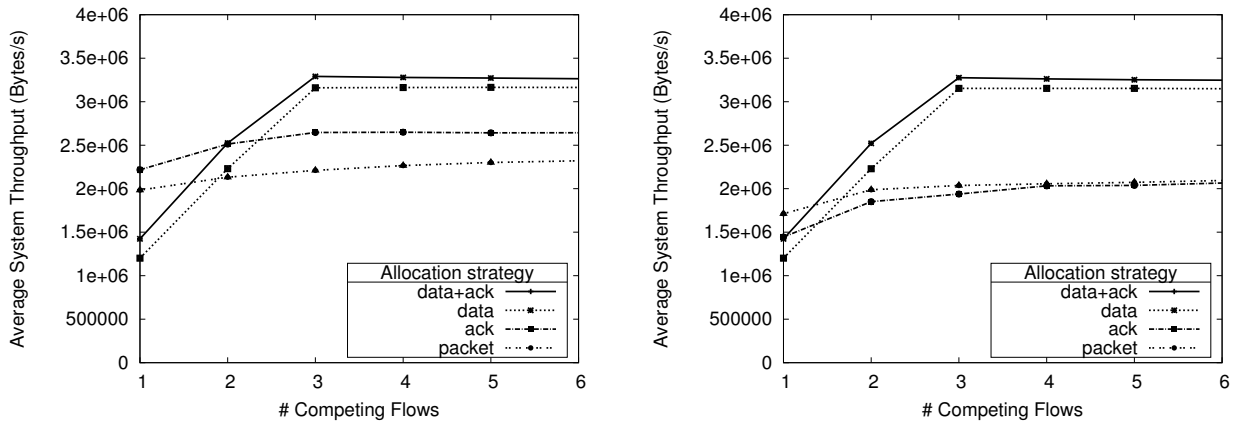
Figure 3 shows the average system throughput (in Bytes/s) for the two reordering mitigations evaluated. Figure 3(a), shows the results for Linux, which performs almost identical to NewReno when (data+ack) and (data) is used. However, for (packet) Linux performs slightly better than NewReno. Furthermore, when (ack) is used, Linux achieves noticeably higher throughput than NewReno. The reason for the performance advantage 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 a situation, TCP is forced to wait for a lengthy retransmission timeout (RTO). When the RTO finally occurs, the threshold is reset to three, which will make TCP vulnerable to unnecessary retransmissions until the threshold has converged again. Second, when the duplicate acknowledgment threshold is increased, TCP will actually allow more reordering to happen, as compared to NewReno, which hurts the performance. The reason why more reordering occurs using mitigation techniques as Linux or NCR, is that the reordering mitigation will help TCP 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.

There are several ways of illustrating the amount of reordering in a particular scenario, including the percentage of reordered packets and/or out-of-order arrivals. Such simple metrics do, however, not contain enough information to reveal both the amount of reordering occurrences and the degree of reordering for each occurrence, i.e., the length distribution of reordering events. As described previously, the length of each reordering event is important as it reveals whether TCP mistakes such an event as packet loss (for lengths of three and more) or not. To obtain this information, we use the reorder density (RD) metric [15], which can be used to construct histograms describing the length distribution of reordering events.

Figure 4 shows RD histograms for both NewReno and Linux, when using (packet). It depicts the fraction of packets arriving in-sequence ($x = 0$) and out-of-order ($x \neq 0$). The negative $x$-values correspond to packets arriving too early and the positive values correspond to packets arriving late. As indicated by the histogram, about $40\%$ of all packets had no displacement and thus arrived in-order when NewReno was used (Figure 4(a)). Furthermore, only a few percent of all packets had displacements of three or more, limiting the amount of unnecessary fast retransmissions.




(a) Linux

(b) NCR

Figure 3. Average system throughput for different reordering mitigation techniques.

Linux encounters much more reordering, as shown in Figure 4(b). Only $15\%$ of all packets are received in-order. This is, as previously mentioned, a consequence of Linux's ability to sustain a larger congestion window during reordering. This leads to more packets being in flight, which in turn increases the amount of reordered packets.

Let us now consider the performance of NCR, which is shown in Figure 3(b). As indicated by the graph, the performance of NCR is very similar to that of NewReno. The reason for NCR's poor performance is related to the design of the scheme. NCR is actually 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. Actually, it exits slow-start (the capacity probing phase of TCP) 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. This, in turn, causes the transmission rate to be constantly low.

## V. Chaska Metropolitan Network Topology

In a real deployment, the topology and connection graph limits the possibilities for multi-path, e.g. for paths located in the edges of a sparsely connected network. Therefore, to evaluate this mix of both multi- and single-path opportunities, we used a a larger and more realistic topology based on a subset of a commercial WMN, deployed in the town of Chaska Minnesota [16].

### A. Simulation Setup

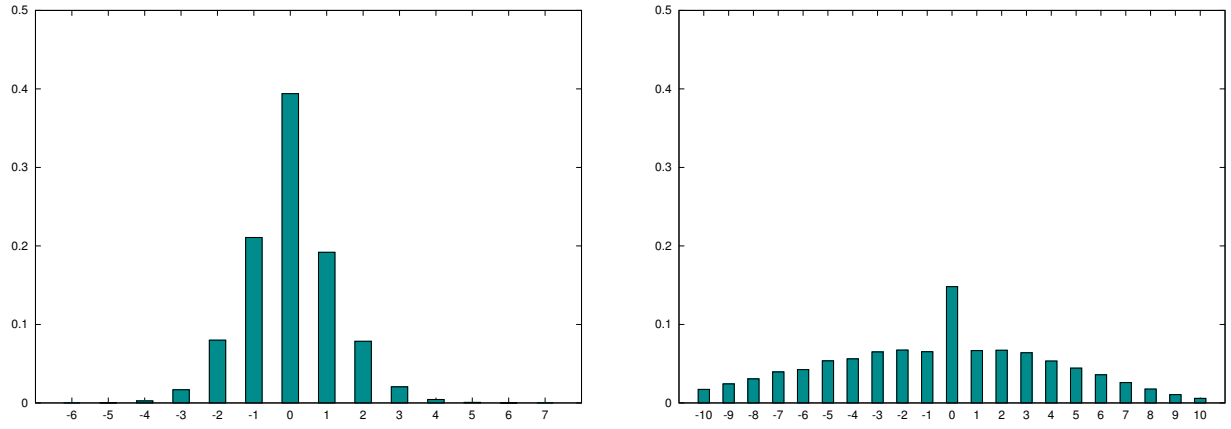
We used the same basic setup, allocation strategies and TCP versions as in the previous simple experiment. The topology used was, however, very different and is illustrated in Figure 5. We also further relaxed the constraint of orthogonal channels between nodes so that multiple nodes might share channels. In total seven channels were used and each node, except nodes $B$ and $C$, was equipped with two radios. The channels were assigned according to the FCRA algorithm [3], mentioned in Section II, which considers the expected utilization (flow-rate) of each link. The flow-rate of each link was calculated to maximize the system throughput for the given topology. The calculated utilization, of the highest utilized links, is indicated in Figure 5. The thickness of the lines illustrates the depicted load on each link when traffic is sent between all end nodes (as in the cross-traffic experiment listed below).

Furthermore, to account for different realistic network scenarios, the following set of traffic profiles were evaluated:

- SC : A single client ($C$) served by one Internet gateway ($A$).
- MC : Multiple clients ($B$, $C$ and $D$) served by one Internet gateway ($A$).
- CT : A cross-traffic experiment with flows between $A \rightarrow C$, $B \rightarrow D$, $C \rightarrow B$ and $D \rightarrow B$

### B. Results

Figure 6 shows the system throughput for the SC scenario, when three TCP flows are sent from node $A$ to node $C$. In the graph, the average throughput is plotted for each combination of TCP version (Linux, NewReno, NCR) and allocation strategy (data+ack, data, ack, packet). As shown in the graph there is no benefit of using (packet), although the topology contains numerous paths between sender and recevier. The reason for this can be found in the topology; as node $C$ is connected with a single link to the network,




(a) NewReno (b) Linux

Figure 4. Reordering density for packet-based routing.

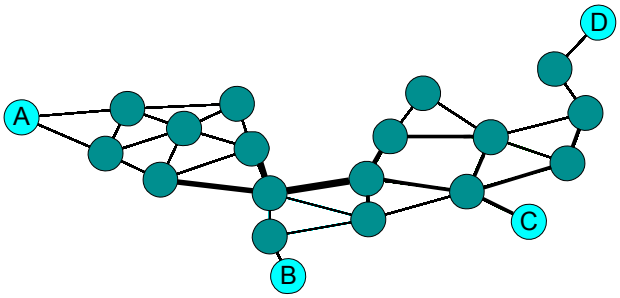



Figure 5. Connection graph of the Chaska subset used for the experiment.

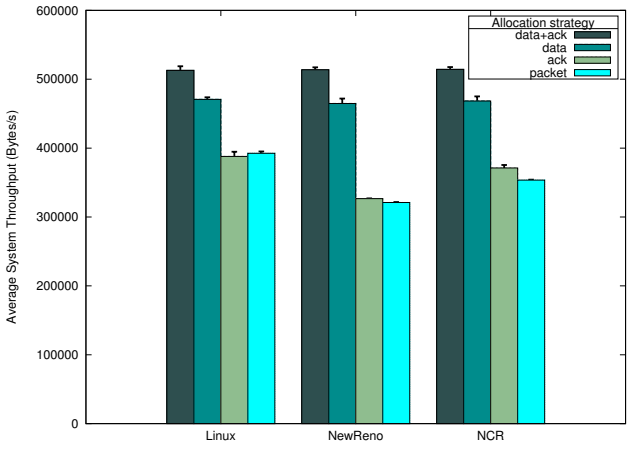



Figure 6. Average system throughput for SC with three flows; 3x($A \rightarrow C$), using different versions of TCP and different allocation strategies.

there is limited benefit of using multi-path forwarding, as all packets need to traverse the same last link. Actually, when not using (data+ack) or (data) the performance is reduced; as packets experiencing reordering have to travel longer paths and still wait at the last hop. However, Linux and NCR are

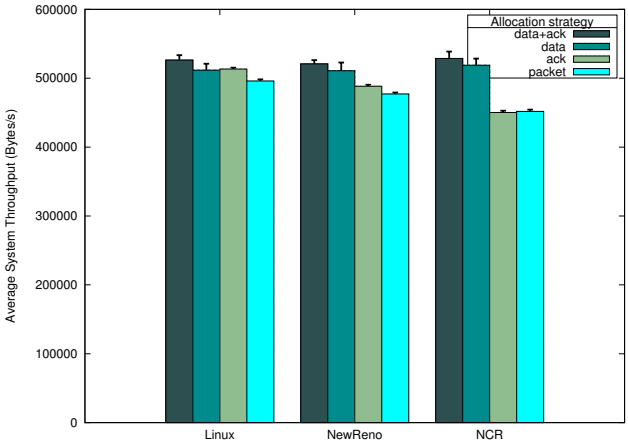



Figure 7. Average system throughput for MC with three flows; $A \rightarrow B$, $A \rightarrow C$ and $A \rightarrow D$, using different versions of TCP and different allocation strategies.

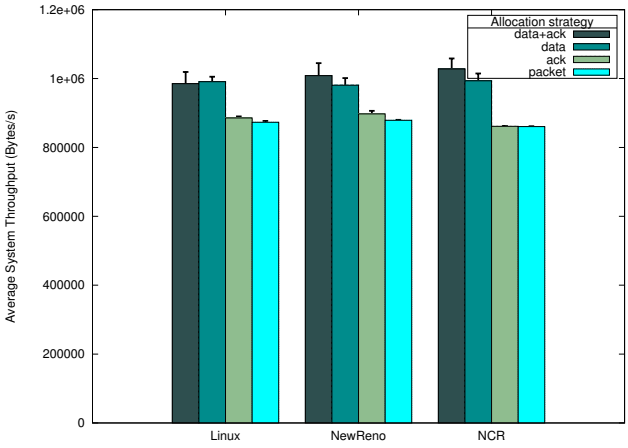



Figure 8. Average system throughput for CT with four flows; $A \rightarrow C$, $B \rightarrow D$, $C \rightarrow B$ and $D \rightarrow B$ using different versions of TCP and different allocation strategies.

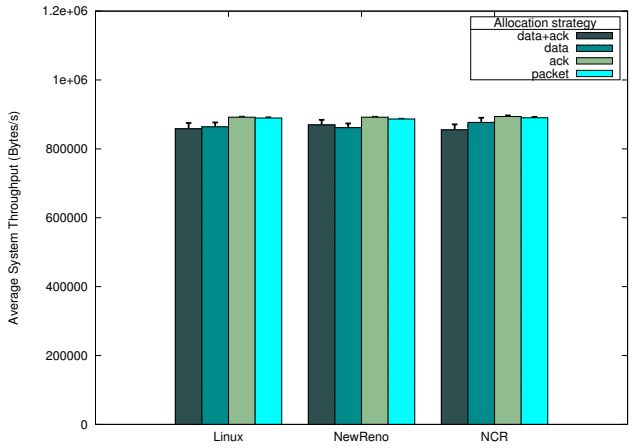



Figure 9. Average system throughput for CT with 24 flows; 6x($A \rightarrow C$), 6x($B \rightarrow D$), 6x($C \rightarrow B$) and 6x($D \rightarrow B$), using different versions of TCP and different allocation strategies.

more tolerant to reordering and can thus sustain a slightly better throughput than NewReno.

When we simulated a gateway serving three different clients (MC) one flow each, see Figure 7, the results were very different from the SC experiment with three flows between one sender and one receiver. Although the number of flows is the same as in the former experiment, the relative benefit of using (data+ack) and (data) allocation is greatly reduced. When flows are destined to three different nodes, there is a greater possibility for (beneficial) path diversity than when flows are destined for only one host. Although the benefit has decreased, it should be noted that it is still beneficial to use (data+ack) or (data) in this experiment.

To evaluate possible effects of cross-traffic, we ran experiments with single flows going between nodes $A \rightarrow C$, $B \rightarrow D$, $C \rightarrow B$ and $D \rightarrow B$ (CT experiment). The results are shown in Figure 8. Consistent with previous results using this topology, (data+ack) and (data) achieves the best performance. Interestingly, the average system throughput have almost doubled, compared to the SC and MC experiments. The reason for this is twofold. When traffic is flowing in both directions to multiple hosts, the network resources can be utilized more efficiently, due to the spatial diversity combined with the benefit of having multiple orthogonal channels. In addition, flow $C \rightarrow B$ has few hops, thus resulting in large throughput gains.

We also increased the number of flows between each node pair in the CT experiment from 1 to 6, making the total amount of flows 24. The results, shown in Figure 9, indicate that (ack) and (packet) are slightly better than (data+ack) and (data). With a large amount of TCP flows sharing the same network capacity, the number of outstanding packets for each flow is much smaller than if there are only a few TCP flows. As we discussed in section IV, when a TCP flow has few outstanding packets there is little possibility for reordering. Therefore, the (negative) impact of packet reordering will be less when the number of flows increases. Furthermore, as discussed earlier, flow-based allocation pins a flow to a certain path. When a large number of flows are served, this can lead to areas in the network where many flows pass through seen as thick lines in Figure 5, creating shared bottlenecks that might hamper performance. However, with (packet) and (ack), such areas will not be created to the same extent, as the amount of traffic that is pinned to a certain path is less and for the majority of the traffic the forwarding decisions are based on the current network status. This problem may be partly ameliorated for (data+ack) and (data) by periodically reassessing the paths to which the flows are assigned.

## VI. Related Work

In addition to the mitigation techniques evaluated in this paper, several others have been proposed. Blanton and Allman [6] have suggested and evaluated a purely reactive mitigation scheme that simply restores the congestion state whenever a retransmission is deemed as unnecessary. Pro-active schemes include e.g. TCP for Persistent Reordering (TCP-PR) [7], where duplicate acknowledgments are not used to detect packet loss which makes it more robust to reordering. Schemes where both reactive and pro-active components are used include, for instance, Reordering-Robust TCP (RR-TCP) [9] where the duplicate acknowledgment threshold is adjusted dynamically. Apart from the evaluations found in the papers describing these mitigations, there have been some independent studies [1], [17], [18]. In [18], the authors show that packet reordering mitigations can improve performance if reordering is the only impairment. When packets are also lost in the network, the performance quickly deteriorates. This is consistent with our findings, as the experiments in the Chaska topology did not reveal a significant performance difference between using a reordering mitigation technique or not. Furthermore, [1] also finds reordered acknowledgments to be a problem for mitigation techniques, as it disrupts the natural feedback loop of TCP.

In this work, we consider a general multi-path routing protocol that requires only minor knowledge about the upper layers, i.e., flow information. In [19], the authors have implemented and evaluated a system called Horizon where the lower layers are tailored to support TCP traffic. As a system approach, Horizon achieves good results; however, due to the use of non-standard headers, middle-boxes could be a problem. Furthermore, as the approach focuses on a scenario where all nodes are located within a single WMN, it is not clear what effects the proposal would have when the sender (or receiver) is located in a wired network.

## VII. Conclusions

In this paper, we have evaluated TCP performance using different multi-path traffic allocation strategies, sometimes

called routing. To get the benefit of load-balancing the traffic over multiple paths while trying to avoid the drawback of packet reordering, we used TCP versions robust to 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 topology with a single flow.

In all other tested settings, TCP throughput was equal or higher with flow-based allocation. The reason for this was simply the lack of paths that were not limited by shared bottlenecks and other areas of interference. In a WMN, with multiple paths of different delay characteristics, packet-based allocation can create a large amount of reordering. When facing this large amount of reordering, the TCP implementations with reordering robustness did not perform optimally, especially when both reordering and packet loss occurred at the same time.

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 [20].

## 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] 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.

[4] 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.

[5] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," *Internet RFCs, ISSN 2070-1721, RFC 5681*, September 2009.

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

[7] S. Bohacek, J. P. Hespanha, J. Lee, C. Lim, and K. Obraczka, "A new TCP for persistent packet reordering," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, April 2006.

[8] Linux Kernel Organization Inc, "The Linux kernel archives," 1997, http://www.kernel.org [accessed 2012.05.16].

[9] 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.

[10] E. Blanton and M. Allman, "Using TCP duplicate selective acknowledgement (DSACKs) and stream control transmission protocol (SCTP) duplicate transmission sequence numbers (TSNs) to detect spurious retransmissions," *Internet RFCs, ISSN 2070-1721, RFC 3708*, February 2004.

[11] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," *Internet RFCs, ISSN 2070-1721, RFC 1323*, May 1992.

[12] 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.

[13] S. McCanne, S. Floyd, K. Fall, K. Varadhan *et al.*, "The network simulator – ns-2," 1995, http://isi.edu/nsnam/ns/ [accessed 2012.05.16].

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

[15] A. Jayasumana, N. Piratla, T. Banka, A. Bare, and R. Whitner, "Improved packet reordering metrics," *Internet RFCs, ISSN 2070-1721, RFC 5236*, June 2008.

[16] Chaska.net, "Residential high speed wireless Internet access," 2011, http://www.chaska.net, [accessed 2012.05.16].

[17] J. Feng, Z. Ouyang, L. Xu, and B. Ramamurthy, "Packet reordering in high-speed networks and its impact on high-speed TCP variants," *Elsevier Computer Communications*, vol. 32, no. 1, January 2009.

[18] D. Yang, K.-C. Leung, and V. O. K. Li, "Simulation-based comparisons of solutions for TCP packet reordering in wireless networks," in *Proceedings of the IEEE Wireless Communications & Networking Conference (WCNC)*, Hong Kong, March 2007.

[19] B. Radunović, C. Gkantsidis, D. Gunawardena, and P. Key, "Horizon: balancing TCP over multiple paths in wireless mesh network," in *Proceedings of the 14th ACM international conference on Mobile computing and networking (MobiCom)*, San Francisco, USA, September 2008.

[20] 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.