Spurious TCP Timeouts in 802.11 Networks

David Malone, Douglas J. Leith, Anshuman Aggarwal, Ian Dangerfield

***Abstract*— In this paper, we investigate spurious TCP timeouts in 802.11 wireless networks. Though timeouts can be a problem for uploads from an 802.11 network, these timeouts are not spurious but are caused by a bottleneck at the access point. Once this bottleneck is removed, we find that spurious timeouts are rare, even in the face of large changes in numbers of active stations or PHY rate.**

I. Introduction

In wired networks, transmission delay is approximately fixed and much of the variability in round-trip times (RTTs) comes from queueing delay. In WiFi networks, retransmissions due to collisions and changing channel conditions are the norm and this additional variable delay has raised concerns for those running TCP over wireless networks. For example, Figure 1 shows a time history of the measured service time (time between enqueueing of packet at WiFi driver and its successful transmission) of a TCP data packet at an 802.11 station. Clearly, this component of the RTT alone is highly variable, and leads to the concern that poor tracking of the RTT could lead to spurious TCP timeouts.

Spurious timeouts have been reported in cellular wireless links [1], but the situation for 802.11 links is less clear. Although 802.11 "hot-spot" operation is ubiquitous, few measurements appear to be available of TCP timeout behaviour in hot-spot/office environments. Previous work on 802.11 links has been confined to the impact of artificial channel impairments e.g. to emulate the impact of mobility and changes in access point (AP) association.

This paper uses experimental measurements to investigate the role of TCP timeouts in 802.11 infrastructure mode WLANs. We show explicitly that the unfairness demonstrated in [2] between flows can be attributed to TCP timeouts, mainly due to TCP ACK losses at the AP buffer, caused by the MAC assigning too few transmission opportunities to TCP ACKs queued at the AP. We also demonstrate that once the AP has sufficient access to the medium to transmit TCP ACKs, TCP timeouts are rare.

II. Related Work

Problems of TCP timeouts in mobile/cellular environments have been studied, for example in [1]. Possible causes for long delay spikes include moving out of radio coverage, handover between cells and yielding to high priority traffic. [3] reports on measurements from a relatively early GPRS setup, where most problems seem to be associated with mobile connections and automatic link adaption. The results of passively

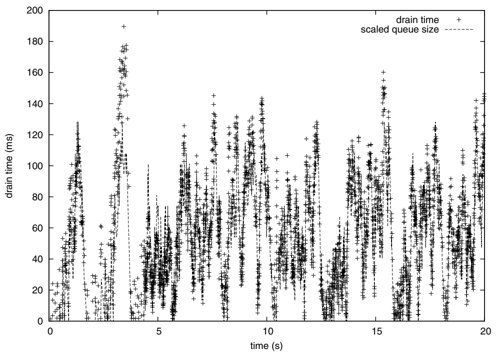


Fig. 1. Measured service time (time between enqueueing of packet at WiFi driver and its successful transmission) of a TCP data packet at an 802.11 station. Four stations uploading through an AP.

monitoring live GPRS/UMTS networks are shown in [4], [5]. Both actually identify a low rate of spurious timeouts in live networks. Techniques have been proposed to reduce spurious timeouts, such as [6] where random delay was artificially added by the network to the RTT was explored as a way to increase the timeout. In [7] this technique was explored in the context of 802.11 networks with bursty outages simulated by dropping n consecutive packets out of every 100, and found to be sometimes beneficial. While these long delays might occur in a network with mobility, it seems that they might be less common in the sort of hot-spot/office environment where 802.11 is often deployed today.

Various modifications to TCP have been suggested that could make it more robust in wireless environments. [8] outlines the Eifel Algorithm, a system for avoiding problems associated with spurious timeouts and spurious retransmits, which can be implemented using the TCP Timestamp header. F-RTO [9] is a different scheme which avoids unnecessary retransmissions after a spurious timeout. Schemes to compensate for other phenomena such as non-congestion loss (e.g. [10]) and sudden changes in bandwidth (e.g. [11]) have also been proposed.

As we will see, timeouts due to the limited number of transmission opportunities of the access point can be important in 802.11. The problems of a slow return path for TCP ACKs is well-known as an issue for TCP [12]. In [2] we demonstrated a simple fix for this problem in WLANs by prioritising TCP ACKs using the 802.11e EDCF.

III. Testbed Setup

The 802.11e wireless testbed is configured in infrastructure mode. It consists of a desktop PC acting as an access point, 15 PC-based embedded Linux boxes based on the Soekris net4801 [13] and one desktop PC acting as client stations.

[1]Work supported by Science Foundation Ireland grant IN3/03/I346 and Cisco Systems. The authors are with the Hamilton Institute, National University of Ireland, Maynooth, Co. Kildare, Ireland.

WIOPT 2008, 1st – 3rd Apr 2008, Berlin, Germany.
Copyright © 2011 – 2012 ICST ISBN 978-963-9799-18-9
DOI 10.4108/ICST.WIOPT2008.3249

Hardware	model	spec
1× AP	Dell GX 260	2.66Ghz P4
15× STA	Soekris net4801	266Mhz 586
1× measurement STA	Dell GX 270	2.8Ghz P4
1× dummynet host	Dell GX 260	2.0Ghz P4
WLAN NIC	D-Link DWL-G520	Atheros AR5212

TABLE I

TESTBED HARDWARE SUMMARY

parameter	value	parameter	value
total buffering	39 packets	MAC Preamble	long
interface tx queue	19 packets	MAC Data rate	11Mbps
driver tx queue	20 packets	MAC ACK rate	11Mbps
dummynet queue	100 packets	MAC Retries	11

TABLE II

TESTBED PARAMETERS SUMMARY

The PC acting as a client records measurements for each of its packets, but otherwise behaves as an ordinary client station. All systems are equipped with an Atheros 802.11b/g PCI card with an external antenna. The system hardware configuration is summarised in Table I. All nodes, including the AP, use a Linux 2.6.8.1 kernel and a version of the MADWiFi [14] wireless driver modified to allow us to adjust the 802.11e CW_{min}, and $TXOP$ parameters. All of the systems are also equipped with a 100Mbps wired Ethernet port, which is used for control of the testbed from a PC. Specific vendor features on the wireless card, such as turbo mode and multi-rate retries, are disabled. All of the tests are performed using the 802.11b physical maximal transmission rate of 11Mbps with RTS/CTS disabled and the channel number explicitly set. Since the wireless stations are based on low power embedded systems, we have tested these wireless stations to confirm that the hardware performance (especially the CPU) is not a bottleneck for wireless transmissions at the 11Mbps PHY rate used. As noted above, a desktop PC is used as a client to record the per-packet statistics. This is to ensure that there is ample disk space, RAM and CPU resources available so that collection of statistics not impact on the transmission of packets. The configuration of the various network buffers and MAC parameters is detailed in Table II.

Note that we use a buffer size of 39 packets, which is smaller than the default for the MadWiFi driver. There are two main reasons for this. First, as we will see, it is relatively easy to end up with round trips of 200m to above 1000ms with this modest buffer size. The upper end of this scale is likely to result in performance that is unacceptable to users for reasons other than TCP timeouts. Increasing buffering only adds to these problems. Second, by reducing the buffer size we may observe more TCP congestion epochs, thus giving a clearer picture of TCP's behaviour over a number of timescales.

The topology used is shown in Figure 2. We generate TCP uploads using Iperf, which run from the stations on the left to the host running dummynet [15] on FreeBSD 6.2. The AP and stations are configured as an infrastructure mode network. The AP is connected to the dummynet host by 100Mbps ethernet. Dummynet is used to emulate propagation delays in the wired path. In the experiments presented here an extra 10ms of delay

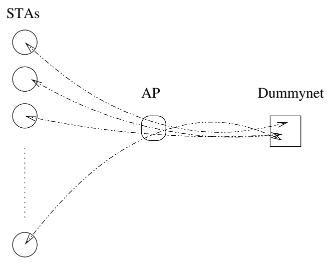


Fig. 2. Topology of testbed.

is introduced as packets enter and leave the dummynet host, for a total of 20ms extra RTT. We have explored other values and seen similar results.

The testbed is not in isolated radio environment, and is subject to the usual impairment seen in an office environment. However, the channel has been selected to be orthogonal to other active 802.11 networks that are nearby.

Several software tools are used within the testbed to generate network traffic and collect performance measurements. While many different network monitoring programs and wireless sniffers exist, no single tool provides all of the functionality required and so we have used a number of common tools including tcpdump[16]. We also used the Linux TCP Probe module [17] to record estimates of round-trip times on the TCP receiver. Network management is carried out using ssh over the wired network.

We will consider the network delay associated with winning access to transmission opportunities in an 802.11 WLAN. This is split into the queueing delay and the MAC access delay (associated with the contention mechanism used in 802.11 WLANs). The MAC layer delay, i.e. the delay from a packet becoming eligible for transmission (reaching the head of the hardware interface queue) to final successful transmission, can range from a few hundred microseconds to hundreds of milliseconds, depending on network conditions. We used the techniques similar to those in [18] to measure MAC delay and queueing delay.

It is interesting to note that for some experiments with larger numbers of stations, network contention is sufficient to make ARP an unreliable protocol. This is because ARP depends on broadcast packets, which are not acknowledged in 802.11, and are consequently not retransmitted at the MAC layer. When the collision probability becomes high, if the ARP tables are not initialised, then it is possible that all ARP level retransmissions will fail and consequently it will be impossible to establish a TCP connection. As we are focusing on TCP issues, we used static ARP entries to avoid this problem. However, this does raise concerns about the performance of ARP in a congested 802.11 environment.

IV. WITHOUT PRIORITISED TCP ACKS

As described in [2], when there are n active TCP uploads sending data packets from different stations in an infrastructure mode network, the AP must return $n/2$ TCP ACKs to the stations (assuming delayed ACKing). On average, the MAC distributes transmission opportunities evenly when the network

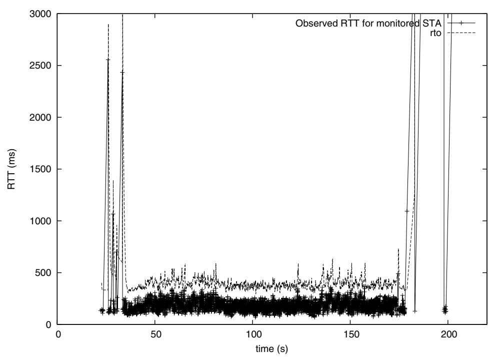


Fig. 4. RTTs for a flow with F-RTO enabled. Four upload flows.

is saturated, so when $n > 2$ there is queueing and loss of TCP ACKs at the AP.

Consider a situation with 4 TCP uploads beginning at the same time, each on a different station. Figure 3(a) shows the observed RTTs at one station. Two different instances are shown. On the left we see that the station has been unlucky and lost some packets early on, it experiences a TCP timeout and never recovers over the 120s of the experiment. On the right, the station has no problems.

Figure 3(b) shows the corresponding queue lengths observed at the access point in both these experiments. We see that, for practical purposes, the queue at the access point is always full, resulting in substantial losses of ACK packets. In both experiments, few other losses are seen in the system. No packets are dropped at the station due to excessive MAC retries and less than a tenth of a percent are dropped at the access point due to excessive MAC retries. In fact, few packets see more than five retries. Thus, the timeouts we observe in this case are not spurious: no matter now long the station waits, it will not receive the TCP ACK that it is expecting. The timeouts are in the class of timeouts related to the loss of an entire flight of TCP ACKs.

Figure 3(a) also shows a curve comparing an estimated round-trip timeout (RTO) to the actual RTT observed. We calculate this timeout using the traditional smoothed round-trip time, $\mathrm{srtt} \leftarrow 7/8\mathrm{srtt} + 1/8\mathrm{rtt}$, and variability $\mathrm{rttvar} = 3/4\mathrm{rttvar} + 1/4|\mathrm{delta}|$. The round-trip timeout shown is $\mathrm{srtt} + max(200\mathrm{ms}, 4\mathrm{rttvar})$. This estimator is close to the those used in both FreeBSD and Linux, and accounts for delayed acking by allowing at least 200ms. We observe that in the case where the station has been lucky and has a reasonable number of packets in flight, it is not timing out, despite the substantial loss of TCP ACKs.

We note that the loss of TCP ACKs can be so severe that even an advanced timeout recovery mechanism such as F-RTO cannot gracefully deal with the issue. In Figure 4, we see the round-trip times of a flow with F-RTO enabled. The flow initially stalls because the SYN ACK packet is lost, this is followed by several small timeouts (where F-RTO should improve performance) followed later by a long timeout.

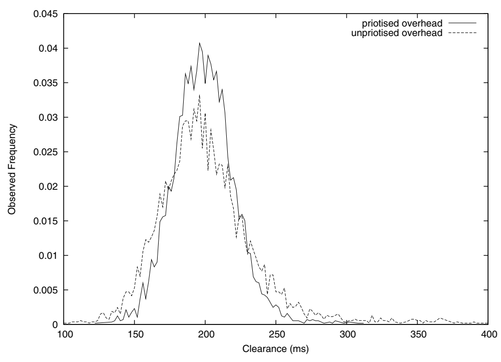


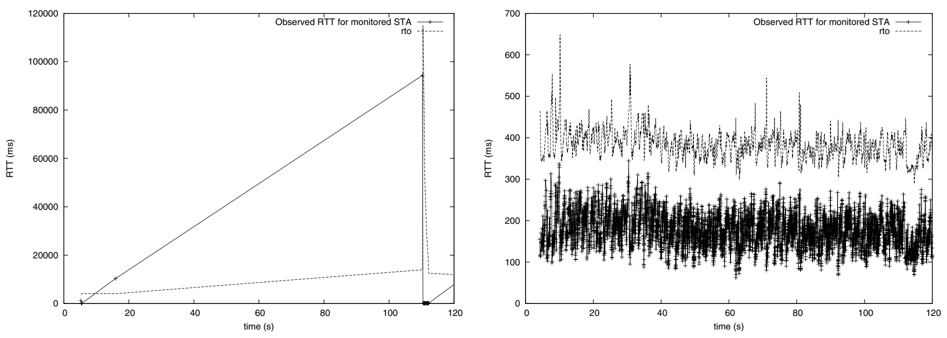
Fig. 6. Observed distribution of clearance between RTO and RTT with and without TCP ACK prioritisation.

V. With prioritised TCP ACKs

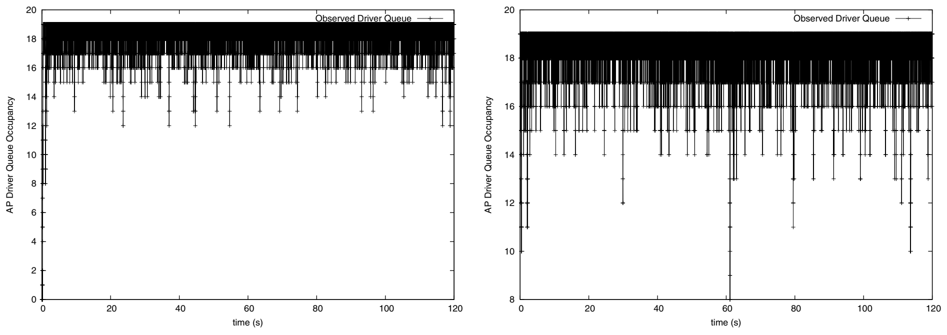
In this section, we prioritise the transmission of TCP ACKs using 802.11e functionality. We increasing the AIFS of TCP data transmited by stations by 4 slots and decrease the CWmin of the TCP ACKs transmitted by the AP to 4. In [2] we saw that this prioritisation resulted in much fairer sharing of bandwidth between TCP uploads. Note that the parameters of this ACK prioritisation scheme are fixed and do not depend on the number of uploads. We again consider 4 uploading TCP flows without F-RTO. Figure 5 shows the typical behaviour.

The queue occupancy at the AP, shown on the right of Figure 5, now remains short; the queue never overflows and rarely has more than 10 packets in it. Losses of packets due to excessive MAC retries remain at a similarly low level as before. The round-trip times shown on the left are still noisy. However, compared to Figure 3, there may be slightly more clearance between the observed round-trip times and the round-trip timeout. Figure 6 shows the observed distribution of $\mathrm{RTO} - \mathrm{RTT}$. Though the means are similar, without TCP ACK prioritisation the distribution is broader with longer tails, confirming more variable behaviour in this case. Note that neither distribution gets close to negative overhead, suggesting that spurious timeouts situations should be rare in either situation.

Even though we see no spurious timeouts in practice, it is interesting to consider the reason for this difference in variability. Some of the spikes present in Figure 3 may be caused by bursts of ACK losses, and these would not be present in Figure 5. The shorter queue occupancy may also make the RTT easier to track: the contribution of ACK queueing to the round-trip time in the first case is the sum of twenty random service times, whereas it is the sum of a considerably smaller number when ACKs are prioritised because queue occupancy is lower. This is illustrated in figure 7, which plots the drain time for TCP ACK packets arriving at the AP is both without and with ACK priorisation (on the left and right respectively). Unsurprisingly, the drain times are much shorter when ACKs are prioritised, the range of the prioritised graph is sixteen times smaller than the prioritised one. Their variability is reduced too. We also show the queue length, scaled to be on a similar range to the drain time. In both cases queue length is correlated with drain



(a) RTTs observed by a TCP flow when ACKs are not prioritised. On the left, the flow repeatedly times out. On the right, the flow manages to keep packets in flight.



(b) AP driver queue occupancy when ACKs are not prioritised. The queue is full of TCP ACKs at most times.

Fig. 3. Measurements of RTT and queue occupancy. Four upload flows.

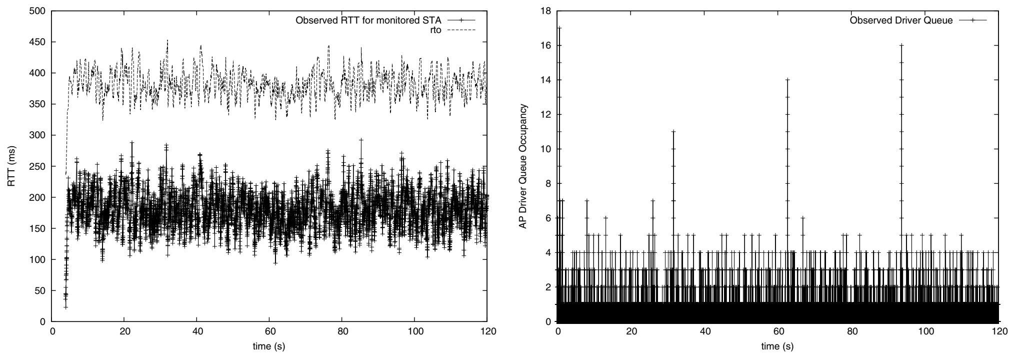


Fig. 5. RTTs for a flow and AP driver queue occupancy when ACKs are prioritised. Four upload flows.

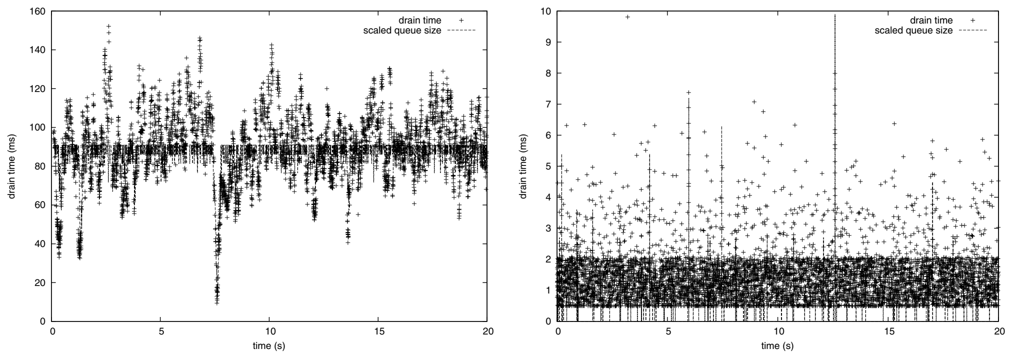


Fig. 7. Drain time for ACKs at the AP. On the left we have no TCP ACK prioritisation, on the right TCP ACKs are prioritised.

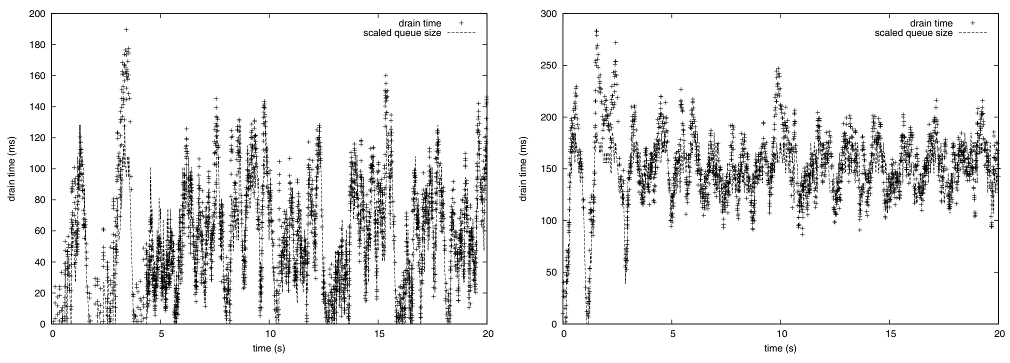


Fig. 8. Drain time for data packets at a station. On the left we have no TCP ACK prioritisation, on the right TCP ACKs are prioritised.

time, but there is substantial noise due to the random MAC service. The corresponding drain time measurements for the data stations are shown in Figure 8. As we expect, the drain time for the data stations increases when we prioritise TCP ACK packets at the AP. With fewer timeouts, TCP flows can grow their cwnd to fill the interface buffer.

While the foregoing measurements are for a single network condition (four upload flows), we observe very similar behaviour across a wide range of conditions.

VI. Impact of Changing Traffic Conditions

The foregoing section considers a fixed network condition (number of flows, link rate etc). In this section we consider the impact of changes in network condition on TCP timeout behaviour. We begin by considering changes in the number of active stations. In 802.11, the available service time is shared between the active stations. If, for example, the number of active stations doubles, then the available service per station roughly halves, and so the apparent time to serve a packet should approximately double. This change in service time could have a significant impact on the queueing delay for both TCP data packets and ACKs, resulting in a large change in round-trip time.

Consider an infrastructure mode network with ACKs prioritised as described in Section V. We begin with 4 flows, and at 120s we add either 4, 8 or 12 flows. We monitor the RTTs of one flow. F-RTO is enabled. As we can see from Figure 9, there is a substantial change in the observed RTT from around 200ms (4 flows) to about 400ms (4+4 flows), 600ms (4+8 flows) and 900–1000ms (4+12 flows), so the RTT is increasing roughly as we expect.

The collision probability is also increasing (the monitor station observes 18% for 4 stations to 35% for 8, 47 % for 12 and 54% for 16 stations), which will be a source of additional delay beyond the linear increase that we expect and increased variability of round trip times. Even with these high collision probabilities, less a tenth of a percent of packets are dropped at either the station or access point.

Does this sudden change of network conditions cause any problems for TCP ? Figure 10 shows a close up of the RTT history for one flow just after the additional flows are added. We see that the RTO threshold adapts quickly and has enough slack that it does not actually ever fall below the observed RTT, though it does come close at a number of points. Note that the RTT samples seem widely spaced on this scale, but this is to be expected, as the RTT is a substantial fraction of one second.

There are no timeouts in the measurements shown. There is one instance where cwnd is reset in the 4+4 experiment just before the packet at 128.86s. This is caused by a packet which is lost because the interface queue is full, and then the later retransmitted packet is also lost because the interface queue is again full. We see that the flow recovers promptly from these losses.

Another source of variability in wireless networks that is not usually present in wired networks is a change in the physical rate used to transmit data. The time to transmit a 1500 bytes of payload increases by roughly 11ms when the PHY rate drops from 11Mbps to 1Mbps. Figure 11 shows a similar test to that shown above, but instead of introducing extra flows, we now change the rate used to transmit packets at all stations at 120s. From left to right we show 11Mbps→5.5Mbps, 11Mbps→2Mbps and 11Mbps→1Mbps. The changes in the RTT's mean and variability are immediately obvious.

Again, there are no timeouts in our measurements, as can be confirmed by the close up shown in Figure 12. In the experiment where the rate was dropped to 1Mbps, note that there is an event that causes a noticeable gap in the RTT history at 140s. In fact, this event is not a timeout, but a loss. Inspection of the packet trace reveals that the retransmission is driven by duplicate ACKs.

All stations changing rate at the same time is an unusual event. It seems unlikely that the rate adaption schemes operating on each station might all decide to change rate at the same time. Changing from 11Mbps→1Mbps is also particularly severe, as rate adjustments are usually by no more than a factor of two. However, we see that the TCP RTO estimation is able to deal with this eventuality.

VII. Conclusion

Though TCP timeouts can be a problem in 802.11 networks, in our setup these timeouts are not spurious, but instead due to congestion on the ACK path. Even if we introduce sudden increases in RTT, by adding stations to the network or making large changes to the PHY rate, TCP's retransmission mechanisms seem to be able to adapt.

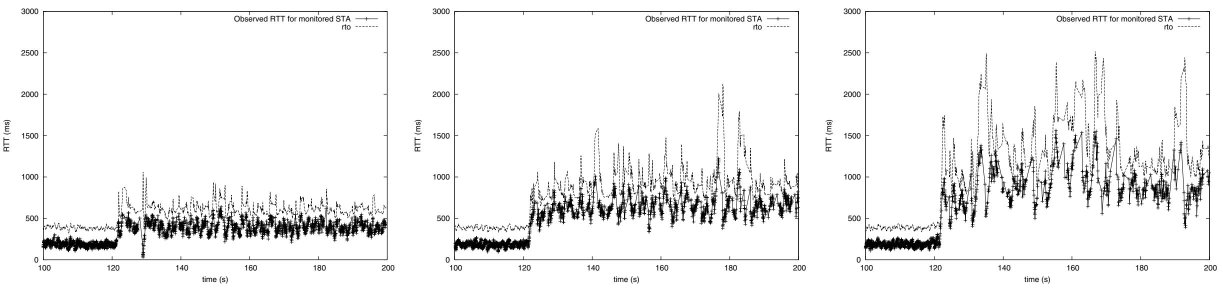


Fig. 9. Beginning with 4 flows, the impact on RTT of adding 4, 8 and 12 flows.

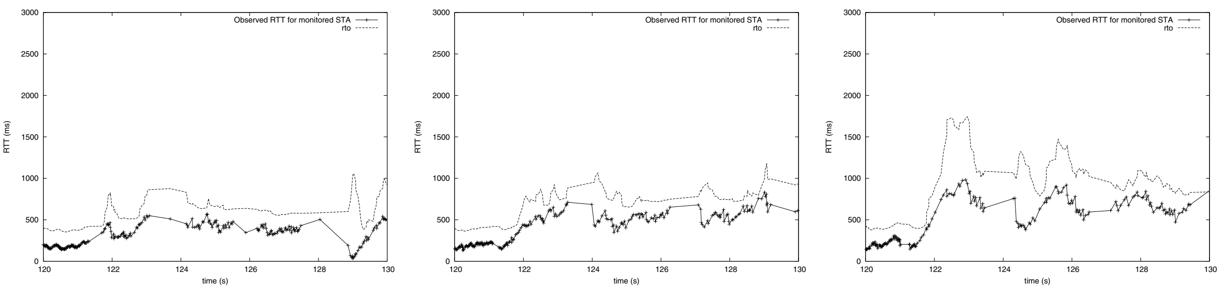


Fig. 10. Close up of the impact on RTT of adding 4, 8 and 12 flows.

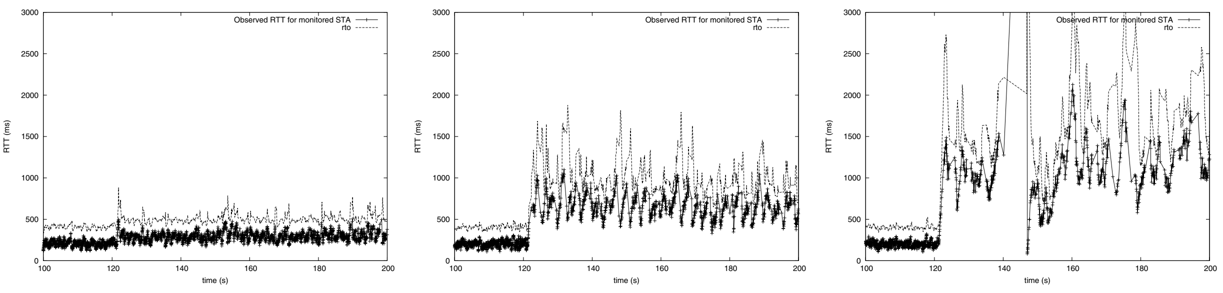


Fig. 11. Beginning with 4 flows with a PHY rate of 11Mbps, the impact of switching to 5.5Mbps, 2Mbps and 1Mbps.

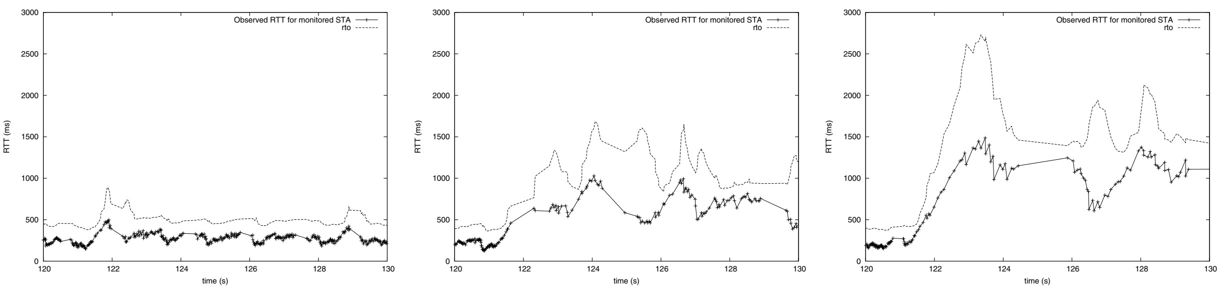


Fig. 12. Close up of the impact of switching to 5.5Mbps, 2Mbps and 1Mbps.

We would also like to expand this work in several directions. We have focused on combinations of TCP uploads, as we expect this situation to be most challenging for 802.11 and TCP. We expect downloads to behave well, because of the absence of a bottle neck on the ACK path. Combinations of uploads and downloads may be interesting, because uploads can out-compete downloads unless measures are taken to restore fairness [2].

As we noted earlier, we have used buffer settings that limit the overall RTT to levels which users are likely to find acceptable. It could be interesting to consider the behaviour of the RTO estimators as buffer size grows. We also note that buffer sizing for TCP in WLANs is an interesting question in itself [19], and is not subject to the usual rules of thumb because of the long term and short term variations in transmission times that we have seen in this paper. The investigation in this paper has been limited to long-lived flows, and further investigation of short-lived flows is planned, as completion times of request-response traffic is important to also consider.

REFERENCES

[1] A. Gurtov, "Making TCP robust against delay spikes," University of Helsinki, Tech. Rep. C-2001-53, November 2001.

[2] A. Ng, D. Malone, and D. Leith, "Experimental evaluation of TCP performance and fairness in an 802.11e test-bed," in *ACM SIGCOMM Workshops*, 2005.

[3] J. Korhonen, O. Aalto, A. Gurtov, and H. Lamanen, "Measured performance of GSM, HSCSD and GPRS," in *Proc. IEEE ICC*, 2001.

[4] F. Vacirca, T. Ziegler, and E. Hasenleithner, "An algorithm to detect TCP spurious timeouts and its application to operational UMTS/GPRS networks," *Computer Networks*, vol. 50, no. 16, pp. 2981–3001, 2006.

[5] F. Ricciato, F. Vacirca, and P. Svoboda, "Diagnosis of capacity bottlenecks via passive monitoring in 3G networks: An empirical analysis," *Computer Networks*, vol. 51, no. 4, pp. 1205–1231, 2007.

[6] T. E. Klein, K. K. Leung, R. Parkinson, and L. G. Samuel, "Avoiding spurious TCP timeouts in wireless networks by delay injection," in *Proc. IEEE GLOBECOM*, November 2004.

[7] G. Fotiadis and V. Siris, "Improving TCP throughput in 802.11 WLANs with high delay variability," in *Proc. ISWCS*, 2005.

[8] R. Ludwig and R. H. Katz, "The eifel algorithm: making TCP robust against spurious retransmissions," *ACM/SIGCOMM Computer Communication Review*, vol. 30, no. 1, 2000.

[9] P. Sarolahti, M. Kojo, and K. Raatikainen, "F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts," *ACM/SIGCOMM Computer Communication Review*, vol. 33, no. 2, 2003.

[10] V. Subramanian, K. K. Ramakrishnan, S. Kalyanaraman, and L. Ji, "Impact of interference and capture effects in 802.11 wireless networks on TCP," in *Proc. WiTMeMo*, August 2006.

[11] J. Zhang, J. Korhonen, S. Park, and D. Pearce, "TCP quick-adjust by utilizing explicit link characteristic information," in *To appear PAEWN*, 2008.

[12] T. Lakshman, U. Madhow, and B. Suter, "Window-based error recovery and flow control with a slow acknowledgement channel: a study of TCP/IP performance," in *Proc. IEEE INFOCOM*, 1997.

[13] "Soekris engineering," http://www.soekris.com/.

[14] "Multiband Atheros driver for WiFi (MADWiFi)," http://sourceforge.net/projects/madwifi/, r1645 version.

[15] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM/SIGCOMM Computer Communication Review*, vol. 27, no. 1, 1997.

[16] V. Jacobson, C. Leres, and S. McCanne, "tcpdump," http://www.tcpdump.org/.

[17] "TCP probe," http://www.linux-foundation.org/en/Net:TcpProbe.

[18] I. Dangerfield, D. Malone, and D. Leith, "Experimental evaluation of 802.11e EDCA for enhanced voice over WLAN performance," in *International Workshop On Wireless Network Measurement (WiNMee)*, 2006.

[19] T. Li and D. Leith, "Buffer sizing for TCP Flows in 802.11e WLANs," *IEEE Comms Let*, To Appear.