Some Modeling and Estimation Issues in Control of Heterogeneous Networks*

Krister Jacobsson†, Niels Möller, Karl Henrik Johansson and Håkan Hjalmarsson
Department of Signals, Sensors and Systems, Royal Institute of Technology
100 44 Stockholm, Sweden
{krister.jacobsson|nisse|kallej|hjalmars}@s3.kth.se

Abstract

Heterogeneous communication networks with their variety of application demands, uncertain time-varying traffic load, and mixture of wired and wireless links pose several challenging problem in modeling and control. In this paper we focus on the round-trip time (RTT), which is a particularly important variable for efficient end-to-end congestion control. Based on a simple aggregated model of the network, an algorithm combining a Kalman filter and a change detection algorithm is proposed for RTT estimation. It is illustrated on real data that this algorithm provides estimates of significantly better accuracy as compared to the RTT estimator currently used in TCP. We also analyze how wireless links affect the RTT distribution. Link re-transmissions induce delays which do not conform to the assumptions on which the transport protocol is based. This causes undesired TCP control actions which reduce throughput. A link layer solution based on adding carefully selected delays to certain packets is proposed to counteract this problem.

1 Introduction

Congestion control is one of the key components that has enabled the dramatic growth of the Internet. The original idea [11] was to adjust the transmission rate based on the loss probability. The first implementation of this mechanism, denoted TCP Tahoe, was later refined into TCP Reno. This algorithm (together with some of its siblings) is now the dominating transport protocol on the Internet. The throughput and delay experienced by individual users are depending on several factors, including the TCP protocol, link capacity and competition from other users. As illustrated in Figure 1, there are also lower layers that may affect the achieved delay and bandwidth, particularly if part of the end-to-end connection is a wireless link. Standard TCP protocols encounter difficulties when wired and wireless links are mixed, since packet loss and packet delays on wireless links may be interpreted as congestion by TCP. Several approaches to counteract this problem has been suggested in the literature. Modifications of TCP has been proposed [20, 26]. Other methods try to more directly differentiate loss as being either due to congestion or due to lossy wireless transmissions [5, 25, 9]. Performance-enhancing proxies is an alternative in which either split connection schemes or interception schemes are used. The first approach introduces a virtual user at the link which acts as receiver to the source and source to the receiver. In the latter approach, acknowledgments are monitored and dropped if they indicate packet loss due to link-level retransmissions. Finally, it is possible to counteract the influence from the wireless link by letting the receiver control the transmission via its advertised window. See [7, 22] for further details on these schemes.

The outline of the paper is as follows. In Section 2 a brief presentation of TCP, and especially the relation to RTT, is given. This motivates the RTT model and estimation scheme presented in Section 3. A new algorithm based on a Kalman filter and change detection [10] is proposed for RTT estimation. In scenarios where new cross-traffic flows cause bottle-neck queues to rapidly build up, the algorithm is shown to be particularly useful to track the rapid changes of the RTT. It gives significantly better accuracy compared to the RTT

*This work was supported by European Commission through the project EURONGI and by Swedish Research Council.
†Corresponding author

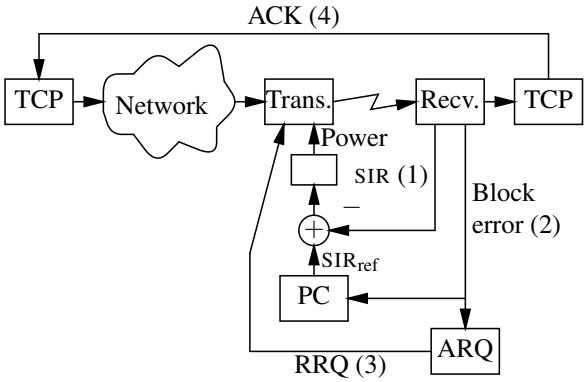

Figure 1: End-to-end congestion control is affected by the delay and the bandwidth of the wired part of the network, but also by feedback mechanisms in lower layers of the wireless links.

estimator currently used in most TCP versions. There are studies of the statistics of network quantities [1, 2] and even estimation of RTT at the link level [12]. However, there seems to be minor work on considering RTT estimation from the perspective of TCP. In Section 4, we study when lost packets are retransmitted locally as in the up-link traffic in today's mobile networks. We show that the local power control and local link retransmission mechanisms interact with TCP in that the delay distribution may be significantly different from the assumptions in TCP. This difference triggers spurious timeouts and thus reduces throughput. We propose a simple solution to this problem in which the delay distribution is shaped in the link itself, before packets are re-transmitted. Another contribution in this direction is [24].

2 TCP and RTT

TCP is window-based which means that each sender has a window that determines how many packets in flight that are allowed at any given time. The transmission rate is regulated by adjusting this window. For a network path with available bandwidth b and RTT τ, the optimal window size is $b\tau$, in the sense that if all users employed this window there would be no queues and the link capacities would be fully utilized. Using loss probability (as in TCP) to measure congestion means that the capacity of the network cannot be fully utilized. With necessity, queues have to build up (causing increased delays) and queues have to overflow (causing loss of throughput). Several methods to cope with these short-comings have been suggested. In TCP Vegas [4] a source tries to estimate the number of packets buffered along its path and regulates the transmission rate so that this number is low (typically equal to three). One interpretation of this algorithm is that it estimates the round-trip queuing delay and sets the rate proportional to the ratio of the round-trip propagation delay and the queuing delay [18]. Both round-trip delays are obtained from measurements of the RTT.

Congestion can be indicated in more sophisticated manners than just dropping packets. In random early detection (RED) [8], packets are dropped before the buffer is full, with a probability that increases with the queue length. RED can thus be seen as a way of indirectly signaling the queue length to the source. In explicit congestion notification (ECN) the links add information concerning their status to the packets. As there is only one bit available in the packet header for ECN, clever coding is required, e.g., random exponential marking (REM) [3].

The transmission rate control problem can be solved in a completely decentralized manner, as was recently shown in [13, 14]. Each source has a (concave) utility function of its rate. The optimization problem is to maximize the sum of the utility functions for all sources. It is shown that in order to solve this problem each source needs to know the sum of the link prices on the path. The link price is a penalty function corresponding to the capacity constraint. It is a function of the total arrival rate at the link and can thus be computed locally at each router. This optimization perspective of the rate control problem has been taken in a number of contributions. The developed algorithms can be classified as (1) primal, when the control at the source is dynamic but the link uses a static law; (2) dual, when the link uses a dynamic law but the source control is static; and (3) primal–dual, when dynamic controls are used both at the source and the links, see [16, 19] for nice overviews. By appropriate choice of utility function even protocols not based on optimization, such as TCP Reno, can be interpreted as distributed algorithms trying to maximize the total utility [17, 19]. TCP Vegas can be classified as a primal–dual algorithm with the queuing delay as a dynamic link price.

Network state variables such as queueing delays and RTT are essential for efficient congestion control, as has been recognized by many researchers, cf., [23]. This has a simple intuitive interpretation: The ideal window

is $b\tau$, where b is available bandwidth and τ the RTT. Hence, the more accurate estimates of b and τ that are available, the closer to the ideal situation it is possible to keep each flow. It is also clear that when only such indirect measures of congestion can be used, throughput will suffer somewhat, since queues have to start to build up in order for the source to detect an increased delay. The queues can be seen as a way to smooth out the uncertainty in the bandwidth and RTT estimates. It follows that one can obtain higher throughput than TCP Reno, since TCP Reno fills up the bottle-neck completely before reacting.

3 RTT Estimation

Motivated by the previous discussion on the importance of accurate RTT estimates, we now take a closer look at short-range RTT from a statistical perspective. The raw RTT measurements, obtained from packet acknowledgements (ACK's), include delays caused by transient effects in the network (attributed to short-lived cross-traffic). The short-lived duration of these flows means that their contribution to the RTT can be considered as noise from the point of view of congestion control. It is thus reasonable to filter them out. In present versions of TCP this is done with a first-order low-pass filter. The average RTT (averaged over a few RTT) often makes sudden changes due to the appearance of a long-lived cross-flow somewhere along the path. It is then important for the filter to react quickly to this change, since otherwise buffers will start to build up with enlarged risk of packet loss and increased delay as consequences. It is impossible with a first-order filter to rapidly adjust to these changes. This motivates the use of a filter based on change detection for RTT estimation, as presented in this section. Before introducing the algorithm, we present a simple model for RTT. The section is ended by experimental evaluations.

3.1 Model

Let $\tau(t)$ denote RTT at the time the ACK is received at the source node. Introduce $d_{l,i}(t)$ for the link delay of link i, $d_{p,i}(t)$ for the corresponding propagation delay, and $d_{q,i}(t)$ for the queue delay. Suppose the considered end-to-end connection has m nodes. Then,

$$\tau(t) = \sum_{i=1}^{m} \left(d_{l,i}(t) + d_{p,i}(t) + d_{q,i}(t) \right).$$

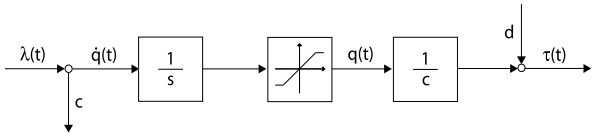

Figure 2: Network node model.

Assume, for now, that the path through the network remains constant during a session. The propagation and link delay is fairly constant and thus only contributes with a bias to the RTT estimate. All the major RTT dynamics is depending on the evolution of the queue states along the path.

The simple dynamics of the queue length $q_i(t)$ at node i is a saturated integrator, as illustrated in Figure 2. Let c_i denote the link capacity and t_i the time instant for the packet arrival at node i. Then, RTT at the time t when the ACK is received is given by

$$\tau(t) = \sum_{i=1}^{m} \frac{q_i(t_i)}{c_i} + \text{bias},$$

where the bias term represents the link and propagation delays. Since queues are 'noisy' due to the burstiness of the arrival traffic, the RTT has a considerable fluctuating characteristic.

3.2 Change detection

The RTT measurements have a high-frequency component that is desirable to detect. To be able to follow step changes in the RTT mean value due to increased network load, new competing traffic flows, or sudden path changes, more advanced algorithms is needed than the first-order linear filter currently implemented in most TCP versions. We propose an adaptive filter with change detection.

Regard RTT as being composed of a smooth desired RTT signal together with an additative high-frequency noise component. We thus model the desired RTT as a noisy observation of a constant exposed to step changes in the mean. If we denote the sampled RTT by y_t and the desired RTT by x_t, we obtain the model

$$\begin{aligned} x_{t+1} &= x_t + \delta_t v_t, \qquad \delta_t \in \{0, 1\} \\ y_t &= x_t + e_t. \end{aligned}$$

The noisy characteristic of RTT is thus captured by the measurement noise e_t with variance R_e. The step

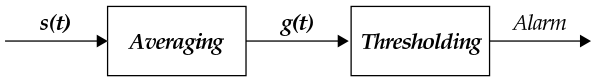

Figure 3: The principle of change detection.

changes in the desired RTT x_t is modeled as the process noise v_t with variance R_v and the discrete variable δ_t. If a change occurs at time t, then $\delta_t = 1$ otherwise $\delta_t = 0$. To estimate the sequence δ^N of instances when a change occurred is a segmentation problem. The principle of change detection is illustrated in Figure 3. The optimal linear estimator is the Kalman filter (assuming Gaussian noise) which is used to estimate x_t.

To estimate the δ^N sequence we use a one-sided cumulative sum (CUSUM) algorithm [10]. Combining the modified Kalman filter with the CUSUM algorithm yields the following adaptive filter, which we denote the CUSUM Kalman filter:

$$\begin{aligned}
\hat{x}_t &= \hat{x}_{t-1} + K_t(y_t - \hat{x}_{t-1}) \\
K_t &= \frac{P_{t-1}}{P_{t-1} + R_e} \\
P_t &= (1 - K_t)P_{t-1} + \delta_{t-1}R_v \\
\epsilon_t &= y_t - \hat{x}_t \\
g_t &= \max(g_{t-1} + \epsilon_t - \xi, 0) \\
&\texttt{if } g_t > h \texttt{ then} \\
&\quad \delta_t = 1 \texttt{ \% alarm} \\
&\quad g_t = 0 \\
&\texttt{else} \\
&\quad \delta_t = 0 \\
&\texttt{end}
\end{aligned}$$

The output of the CUSUM Kalman filter is given by the estimate of the (desired) RTT $\hat{x}_t$. The filter has two design parameters: the negative drift ξ and the alarm threshold h. These parameters are tuned to adjust the sensitivity in the detection procedure. The same values have been used in all our initial experiments with satisfying results, so the filter seems fairly robust. Note that also the variances R_v and R_e influence the filter behavior.

3.3 Experimental evaluation

The objective is to capture the rapid changes that the RTT undergoes when queues build up when for instance the traffic load increases abruptly. We developed a modified ping tool to monitor RTT time series. By sending a dense stream of small packets the RTT is monitored effectively without affecting the network load too much. Note that data is not easily obtained from a TCP session itself, because a rapid traffic load increase would probably trigger the RTO mechanism in TCP (which would mean lack of samples during the transient). Also, if a queue along the path fill up, packet loss occurrences reduces the TCP sending rate with more sporadic time series as a consequence. Recall that the RTT time in congestion control is often in fractions of seconds (typically $10 - 500$ ms).

We measured the RTT between KTH and the CAIDA web site.This path is normally about 20 hops, including the Atlantic link. The mean RTT is approximately 190 ms. The sending interval (sampling time) of the ping tool was 30 ms. According to our measurements, the path is normally not congested with the result that the RTT almost shows a deterministic behavior with a low variance. However, sporadically the traffic load increased with the result of suddenly increased and fluctuating queues, which propagated to the RTT. The proposed CUSUM Kalman filter was used on the collected data sets. The variances R_e and R_v were set to the variance of the RTT samples. The CUSUM design parameters were set to $\xi = 0.005$ and $h = 0.05$ in all experiments. This was done manually based on initial experimentall results.

A detection of a sudden step in the RTT mean value can be seen in Figure 4. The two upper plots show the sampled RTT values, together with the output of the CUSUM Kalman filter and the output of the conventional TCP first-order filter (with gain set to 0.9 [11]). In the lower plot the test statistic g_t is plotted and the change detection alarm is encircled. The CUSUM Kalman filter estimate is smooth, but still manage to detect a mean change as low as 7 %. Note that the TCP filter is more sensitive to noise and periodic fluctuations. As the change in mean is moderate in this example, the TCP filter adapts quite fast. The RTT mean change is probably a result of an abrupt change in the traffic load. An additive static traffic stream as a UDP flow might be the explanation, but a re-routing inside the network is also a possibility.

In Figure 5 we see a large change in the mean RTT of about 100 %. Even if the probing packets are sent with only 30 ms intervals, the RTT measurements do not capture the queue building up. The result becomes a step in the RTT measurements. In this scenario the

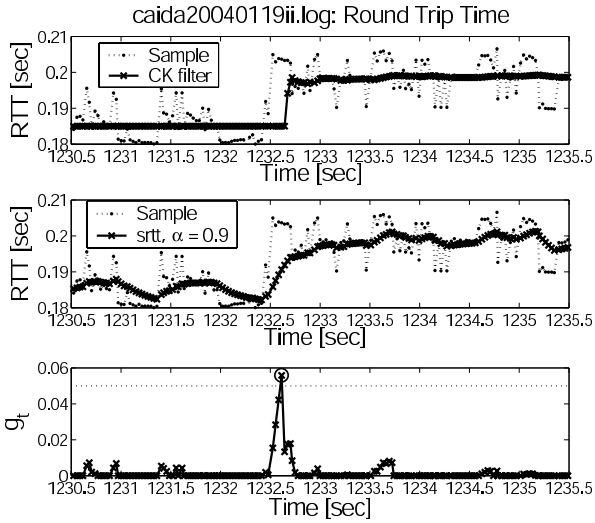

Figure 4: Experimental evaluation of the proposed RTT estimation algorithm on a small change in RTT.

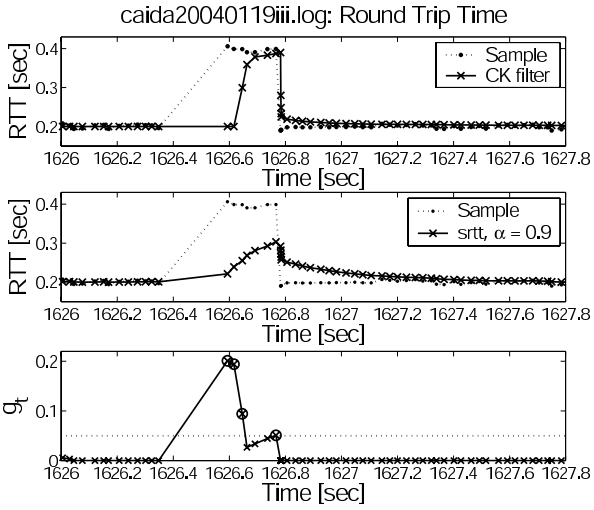

Figure 5: Experimental evaluation of the proposed RTT estimation algorithm on a large change in RTT. Note how well the proposed algorithm (CK filter) captures the rapid change, while the conventional filter in TCP (srtt) adjusts much slower.

proposed RTT estimation algorithm reacts after only a few samples, and is much faster than the filter in TCP. Note that this time it was no static mean change, but the queue vanishes after half a second and go down to its original level. The estimation algorithm reacts on the reset and adapts almost immediately while the TCP filter is lagging.

The two given examples show the main features of the CUSUM Kalman filter: smooth estimates but still fast adaption when drastic changes occur. In an application within a transmission protocol, the sampling time is typically larger and the measurements tend to be more spiky. The probability of hitting a plateau as in Figure 5 with more than a few samples is then low. By filtering out such events the RTT estimate could be kept at the appropriate level.

4 TCP and Link-Layer Interaction

Poor TCP performance over wireless links is a well known problem. In this section, we will try to understand the link properties that are problematic, and try to address them.

The traditional explanation for poor TCP performance is that the wireless links drops packets due to noise on the radio channel, and that TCP interprets all packet losses as indications of network congestion. This explanation is a little too simplistic, when considering wireless links that employ link-layer retransmissions. With such links, we get almost no packet loss, but instead we get random delays, which, it turns out, are also problematic for TCP.

Wireless TCP performance can be attacked at several levels. In this section, we consider link-layer effects. We believe that as far as possible, the link-layer should be engineered to be TCP-friendly, reducing the differences between wired and wireless links. There will naturally be some residual idiosyncrasies of wireless channels that cannot be dealt with in this way; this approach should be viewed as complementing both developments to make TCP more robust to "strange" links, and cross-layer developments that let the link and the end-node TCP:s exchange information about link and flow properties.

4.1 System overview

When using TCP over a wireless link, there are several interacting control systems stacked on top of each other, illustrated in Figure 1. At the lowest level, the transmission power is controlled in order to keep the signal to interference ratio (SIR) at a desired level. This is a fast inner loop intended to reject disturbances in the

form of "fading", or varying radio conditions. On top of this, we have an outer power control loop that tries to keep the block error rate (BLER) constant, by adjusting the target SIR of the inner loop. Next, we have local, link-level, retransmissions of damaged blocks. The outer loop power control and the link-layer retransmissions operate at a time scale of 20 ms, which is the size of the timeslot needed for transmission of a single radio block. Finally, we have the end-to-end congestion control of TCP.

By modeling the lower layers, we can investigate effects the link layer control has on TCP performance. We refer to our previous paper [21] for further details on the radio model.

4.2 Markov chain

The target block error rate is a deployment trade-off between channel quality and the number of required base stations. For UMTS the reference block error rate is often chosen to be about 10%, see [6], which is what we will use.

As there is no simple and universal relationship between the SIR and the block error rate, the outer power control loop uses feedback from the decoding process to adjust SIR_{ref}. The outer loop uses a fix step size Δ. It decreases SIR_{ref} by Δ for each successfully received block, and increases SIR_{ref} by 9Δ each time a block is damaged.

The process can be modeled as a discrete Markov chain, where state k corresponds to $\text{SIR}_{\text{ref}} = k\Delta$. Assuming that the inner loop power control manages to keep the actual SIR close to SIR_{ref}, and using an appropriate channel model, we get a threshold shaped function $f(r)$ which gives the probability of block damage, given $\text{SIR}_{\text{ref}} = r$. Then the Markov chain transitions from state k to state $k+9$, with probability $f(k\Delta)$, and to state $k-1$ with probability $1-f(k\Delta)$. The operating point of the outer loop power control is close to the point where $f(r) = 10\%$, i.e., the desired block error rate.

From $f(r)$ and Δ, it is straight forward to compute the stationary distribution of the Markov chain. Figure 6 shows the stationary distribution for a BPSK channel (see [21] for the parameters) and three different values for Δ.

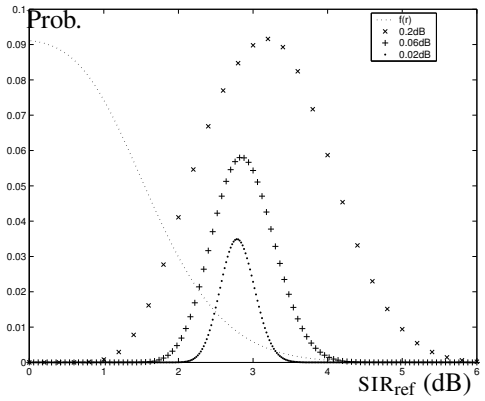

Figure 6: Stationary distribution for the power control. Each mark represents one state of the power control, the corresponding value of SIR_{ref}, and its stationary probability. The dotted curve is the threshold-shaped function $f(r)$, scaled to fit in the figure, which represents the block error probability as a function of SIR_{ref}.

4.3 Link-layer retransmission

Since a packet loss probability on the order of 10% would be detrimental to TCP performance, the link detects block damage (this is the same feedback signal that is used for the outer loop power control), and damaged blocks are scheduled for retransmission. We will consider one simple retransmission scheme, the (1,1,1,1,1)-Negative Acknowledgements scheme [15], which means that we have five "rounds", and in each round we send a single retransmission request. When the receiver detects that the radio block in time slot k is damaged, it sends a retransmission request to the sender. The block will be scheduled for retransmission in slot $k+3$ (where the delay 3 is called the RLP NAK guard time). If also the retransmission results in a damaged block, a new retransmission request is sent and the block is scheduled for retransmission in slot $k+6$. This goes on for a maximum of five retransmissions.

Consider the system at a randomly chosen start time, with the state of the power control distributed according to the stationary distribution. For any finite loss/success sequence (for example, the first block damaged, the next six received successfully, the eighth damaged), we can calculate the probability by conditioning on the initial power control state and following the corresponding transitions of the Markov chain. In the following sections, we use these probabilities to investigate the experience of IP packets traversing the link.

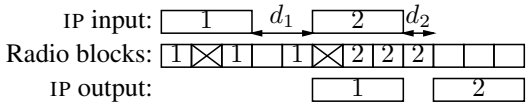

Figure 7: IP packets divided into radio blocks

d	0	40	60	100	120	160	180
p	80.6	8.8	9.3	0.6	0.6	0.03	0.03

Table 1: Delay distribution for a wireless link. The delay d [ms] is the delay due to radio block retransmissions, and p [%] is the corresponding probability.

4.4 IP packet delay

As a link employing link-layer retransmission yields a very small packet loss probability, the most important characteristic of the link is the packet delay distribution. If the distribution is sufficiently "friendly" to TCP, then the layering of the system works nicely, which means that upper layers like TCP need not be aware of any particular properties of individual links in the network.

In this section, we compute the packet delay distribution explicitly from the models described above. Later, we will also assume that the calculated delay probabilities apply independently to all packets, which should be fairly close to reality as long as the power control is working.

When transmitting variable size IP packets over the link, each packet is first divided into fix size radio blocks, see Figure7. We let n denote the number of radio blocks needed for the packet size of interest. For the links we consider, we have $n \leq 10$.

The delay experienced by an IP packet depends on which, if any, of the corresponding radio blocks are damaged, and on the number and scheduling of the block retransmissions. When all blocks are finally received correctly, the IP packet can be reassembled and passed on.

From the probabilities for all possible success/loss sequences at the radio block level, and for each n, we can extract explicit probabilities for the possible IP packet delays. The resulting delay distribution for our example channel, with a power control step size $\Delta = 0.06$dB, and $n = 2$, is shown in Table 1.

The table includes only the delays due to radio block retransmissions, there is also a fix delay of 40 ms for original transmission of two radio blocks.

A TCP timeout event occurs when a packet, or its acknowledgment, is delayed too long. Let RTT_k denote the round-trip time experienced by packet k and its corresponding acknowledgment. The TCP algorithms estimates the mean and deviation of the round-trip time. Let $\widehat{\text{RTT}}_k$ and $\hat{\sigma}_k$ denote the estimated round-trip time and deviation, based on measurements up to RTT_k. TCP then computes the timeout value for the next packet as $R_{\text{TO}} = \widehat{\text{RTT}}_k + 4\hat{\sigma}_k$, which means that the probability that packet k causes a timeout is given by

$$\text{P}(\text{RTT}_k > \widehat{\text{RTT}}_{k-1} + 4\widehat{\sigma}_{k-1}) \tag{1}$$

Timeouts that occur when a packet is severely delayed, but *not* actually lost, are called *spurious timeouts*. A simplified model of TCP is to assume that the the estimation is perfect, and that the timeout value is set to $R_{\text{TO}} = \text{E}(\text{RTT}) + 4\sigma(\text{RTT})$. From the delay distribution of Table 1, we get $R_{\text{TO}} \approx 103$ ms and the probability that the delay is larger is $\approx 0.68\%$. This is the probability of spurious timeout events. When varying the parameters n, Δ, we typically get a probability of spurious timeout on the order of 0.5–1% [21].

4.5 Improving the link-layer?

It is not trivial to define precisely what properties a link should have in order to be friendly to TCP. It seems clear that for example links with normal or uniformly distributed and independent delays are friendly enough. One crude measure is to examine the tail of the distribution. More precisely, if X is a stochastic variable representing the identically and independently distributed packet delays, define

$$\text{P}_{\text{TO}}(X) = \text{P}(X > \text{E}(X) + 4\sigma(X)) \tag{2}$$

The motivation for this measure is the calculation of the timeout value for TCP. Timeout is intended to be the last resort recovery mechanism. For TCP to work properly, a spurious timeout must be a rare event.

Also note that $\text{P}_{\text{TO}}(X)$ is invariant under the addition of constant delays.

For distributions we know are friendly to TCP, P_{TO} is small. For a general distribution, assuming only independence and finite first and second moments, P_{TO} is bounded by Chebyshev's inequality. Comparing these values,

$$X \text{ uniform} \quad \Longrightarrow \quad \text{P}_{\text{TO}}(X) = 0 \tag{3}$$

$$X \text{ normal} \quad \Longrightarrow \quad \text{P}_{\text{TO}}(X) \approx 6.3 \cdot 10^{-4} \tag{4}$$

$$X \text{ wireless} \quad \Longrightarrow \quad \text{P}_{\text{TO}}(X) \sim 100 \cdot 10^{-4} \tag{5}$$

$$X \text{ arbitrary} \quad \Longrightarrow \quad \text{P}_{\text{TO}}(X) = 625 \cdot 10^{-4} \tag{6}$$

we see that the two friendly distributions yield a P_{TO} at least two orders of magnitude below the worst case given by Chebyshev, while the wireless delay yields a significantly higher P_{TO}, although still with some margin to the worst case.

If we want to improve the system, where should we attack it? The power control design have many constraints of its own, relating to radio efficiency and cost of deployment. It seems difficult to design and motivate changes to the power control for improving the delay distribution properties. Improvements to the TCP algorithm in the end-nodes are important, but also difficult both for technical and practical reasons, such as limited information about what goes on in the link (note that the link and the TCP implementations are not only in separate layers, they are also *geographically* separate), and the complex standardization and deployment process.

However, we do have some engineering freedom in the link itself. Even if we do not want to modify the power control, there are other link-local mechanisms we can add or optimize.

- Optimize the retransmission scheduling, taking advantage of the block loss correlation that we get after power control.
- Use error correction coding.
- Tweak the delay distribution by adding additional delays to selected packets.

In the following section, we investigate the simplest of these options, namely the third one.

4.6 Introducing additional delays

Assume that we have a discrete delay distribution for X, $\mathrm{P}(X = d_i) = p_i$, where $d_i < d_{i+1}$. It is typical, but not required, that also $p_i \geq p_{i+1}$. Let μ and σ^2 denote the mean and variance of X.

We consider the following class of tweaks to X. For each packet that experiences a delay $X = d_i$, buffer the packet so that it gets an additional delay x_i. This defines a new distribution $\tilde{X}$, $\mathrm{P}(\tilde{X} = d_i + x_i) = p_i$ (or if it happens that $d_i + x_i = d_j + x_j$ for some $i \neq j$, the corresponding probabilities are added up). For an example of what X and $\tilde{X}$ can look like, see Figures 8 and 9.

We can choose the parameters x_i freely, constrained only by $x_i \geq 0$.

What is the best choice for x_i? We choose a maximum allowed value, ϵ, for $\mathrm{P_{TO}}(\tilde{X})$, and minimize $\mathrm{E}(\tilde{X})$ under the constraint that $\mathrm{P_{TO}}(\tilde{X}) \leq \epsilon$. This means that we want to push down our measure of "TCP-unfriendliness", while at the same time not adding more delay than necessary.

We simplify the problem a little by requiring that $\mathrm{P_{TO}}(\tilde{X})$ should correspond to a tail of the original distribution X. Let k be the smallest value such that $\sum_{i \geq k+2} p_i \leq \epsilon$. Let $c = d_{k+1} + \delta < d_{k+2}$, where δ is a robustness margin. We impose the additional constraints $\mathrm{P_{TO}}(\tilde{X}) = c$, $x_i + d_i \leq d_{k+1}$ for $i \leq k$, and $x_i = 0$ for $i > k$. Then, for any x_i satisfying these new constraints, we will have $\mathrm{P_{TO}}(x) = \sum_{i \geq k+2} p_i \leq \epsilon$. Written as an optimization problem, we have

$$\min \ \mathrm{E}(\tilde{X}) \tag{7}$$

$$\mathrm{P_{TO}}(\tilde{X}) = c \tag{8}$$

$$x \geq 0 \tag{9}$$

$$x_i \leq d_{k+1} - d_i \tag{10}$$

This is a quadratic optimization problem. To write it in matrix form, let x denote the vector $(x_1, \ldots, x_k)^T$, and similarly for p and d. Let $S = 16 \operatorname{diag} p - 17pp^T$, $b_i = 2p_i(16d_i + c - 17\mu)$, $m_i = d_{k+1} - d_i$ and $\alpha = 16\sigma^2 - (c - \mu)^2$, and we can rewrite the problem as

$$\min p^T x \tag{11}$$

$$x^T S x + b^T x + \alpha = 0 \tag{12}$$

$$0 \leq x \leq m \tag{13}$$

Since the symmetric matrix S is typically indefinite, the problem is not convex. But it can be solved in exponential time $O(k^3 3^k)$, which has not been a problem thanks to the very limited size of k.

The typical solution is of the form $x = (0, \ldots, 0, x_j, m_{j+1}, \ldots, m_k)^T$. When the optimum has this form, it means that the cheapest way to to increase $\mathrm{P_{TO}}$, in terms of mean delay, is to increase the x_i corresponding to the smallest p_i. Necessary and sufficient conditions for the optimum to be of this form has not yet been determined.

Now consider the delays d_i and probabilities p_i in Table 1, and assume that packets are independently delayed according to the given probabilities. This distribution is also shown in Figure 8. Before tweaking the delays, we have $\mathrm{E}(X) \approx 10.6$ ms, and $\mathrm{P_{TO}}(X) \approx 0.68$ %.

With $\epsilon = 0.1\%$ and $\delta = 10$ ms, the above optimization procedure yields $k = 4$ and the optimal additional

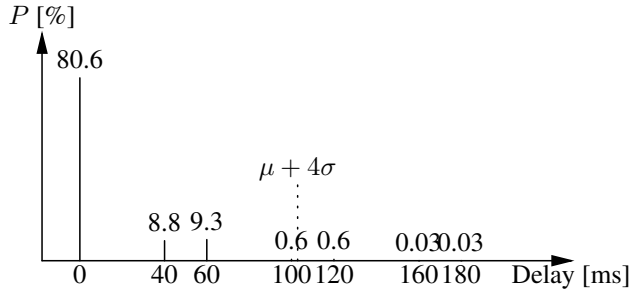

Figure 8: Original delay distribution

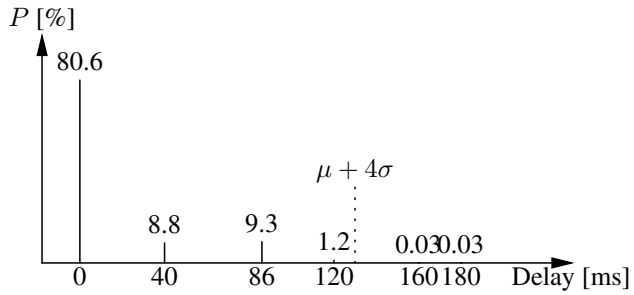

Figure 9: Optimized delay distribution

delay $x \approx (0, 0, 26, 20)^T$ ms. This modified distribution is shown in Figure 9. The mean additional delay is only 2.54 ms, which seems to be a small cost, if we compare it to the transmission delay for the packet, which is 40 ms, or the end-to-end delay which necessarily is even larger. We also achieve $\mathrm{P_{TO}} < \epsilon$, what we actually get is $\mathrm{P_{TO}} \approx 0.06\%$.

4.7 Robustness

The delay seen by the endpoints is the end-to-end delay, which consists of the delay over our link and additional queueing and propagation delays in the rest of the network. We model the delay in the rest of the network as a stochastic variable V (assuming that the sequence of delays are identically and independently distributed). It is then relevant to consider $\mathrm{P_{TO}}(\tilde{X} + V)$. Since $\mathrm{P_{TO}}$ is invariant under the addition of constant delays, we can make the convenient assumption that $\mathrm{E}(V) = 0$.

If we assume that V and $\tilde{X}$ are independent, we can derive a simple bound for $\mathrm{P_{TO}}(\tilde{X} + V)$. First define $c' = \mathrm{E}(\tilde{X} + V) + 4\sigma(\tilde{X} + V)$. Note that $c' \geq c$, so that $c' - x_i - d_i \geq \delta$ for $i \leq k$. Next, we condition on $\tilde{X}$,

$$\begin{aligned}
\mathrm{P_{TO}}(\tilde{X} + V) &= \mathrm{P}(\tilde{X} + V > c') \\
&= \sum_i p_i \mathrm{P}(V > c' - x_i - d_i) \\
&\leq \sum_{i=1}^{k} p_i \mathrm{P}(V > \delta) + \sum_{i>k} p_i \\
&\leq \mathrm{P}(V > \delta) + \epsilon
\end{aligned}$$

This bound shows that if the delay variations in the rest of the network are small enough relative to our robustness parameter δ, $\mathrm{P_{TO}}(\tilde{X} + V)$ will not be much larger than ϵ. For typical distributions, the bound for the first sum is very conservative. This is because the first few p_i dominates, while the corresponding $c' - x_i - d_i$ are significantly larger than δ. A more precise bound can be calculated using additional information about p_i and the distribution of V.

4.8 Performance implications

When computing TCP throughput, there are two distinct cases: Depending on the bandwidth-delay product, throughput can be limited either by the bandwidth of the path across the network, or by the maximum TCP window size.

For a small bandwidth-delay product, a modest buffer before the bottleneck link (which we will assume is our radio link) will be enough to keep the radio link fully utilized. Timeouts, if they occur with a low frequency, will not affect throughput at all. This can be seen for example in the performance evaluation [15]: In the scenarios that have a large maximum window size compared to the bandwidth-delay product, we get a throughput that is the nominal radio link bandwidth times $1 - p$ (where p is the average block loss probability), and there is no significant difference between different link retransmission schemes. Only when bandwidth or delay is increased, or the maximum window size is decreased, do we see a drastic changes in throughput when the BLER or retransmission-scheme varies.

Therefore, we will concentrate on the case of a large bandwidth-delay product. For a concrete example, we will consider the following scenario (see Figure 10: Radio link bandwidth 384 kbit/s, packet size $m = 1500$ bytes, maximum TCP window size $w = 7500$ bytes (i.e. five packets), and a constant round-trip delay time, excluding the radio link itself, of 0.2 s.

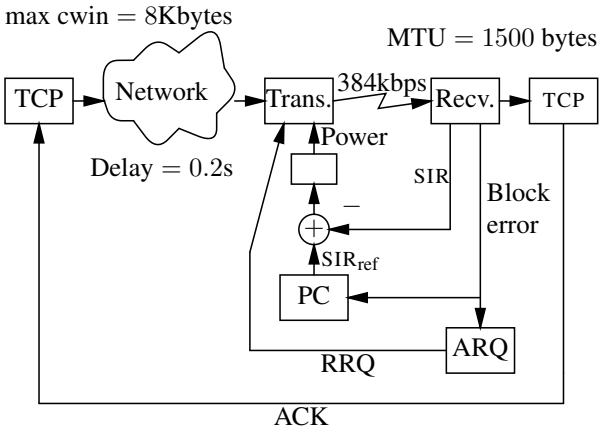

Figure 10: Numerical example

The available radio bandwidth (excluding losses) is 42.2 Kbyte/s. Due to the limited window size, TCP can not utilize the link fully. The ideal TCP throughput is one maximum size window per RTT. For the untweaked link, the mean total RTT is $200+40+10.6 = 250.6$ ms, implying an ideal throughput of 29.2 Kbyte/s.

For each spurious timeout, the sending TCP enters slow start. The window size is reset to 1 packet, and the slowstart threshold is set to 2 packets. For the next four round-trip times, we will send 1, 2, 3, and 4 packets, 10 packets less than if we had kept sending a maximum window of 5 packets every RTT. This leads to the following expression (a more general formula is derived in [21]).

$$\text{Throughput} = \frac{w}{\text{E}(\text{RTT})(1 + 10\text{P}_{\text{TO}})} \qquad (14)$$

Hence, over the untweaked link, we get a throughput of 27.4 Kbyte/s. For the tweaked link, we have a slightly larger RTT (which in itself would decrease the throughput slightly), and a significantly smaller P_{TO}. The resulting throughput is 28.8 Kbyte/s. These figures are summarized in Table 2.

	Kbyte/s
Available radio bandwidth	42.2
Ideal TCP throughput	29.2
With wireless link	27.4
Modified wireless link	28.8

Table 2: Throughput summary

The important point is that a simple but carefully selected modification to the link-layer yields a modest but significant performance improvement.

References

[1] M. Allman. A web server's view of the transport layer. *ACM SIGCOMM Computer Communication Review*, 30(5), 2000.

[2] M. Allman and V. Paxson. On estimating end-to-end network path properties. *ACM SIGCOMM Computer Communication Review*, 31(2), 2001.

[3] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: active queue management. *IEEE Networking*, 15(3):48–53, 2001.

[4] L. S. Brakmo and L. L. Peterson. TCP Vegas: end-to-end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[5] S. Cen, P. C. Cosman, and G. M. Voelker. End-to-end differentiation of congestion and wireless losses. *IEEE/ACM Trans. on Networking*, 11(5):703–717, 2003.

[6] A. Dahlén and P. Ernström. TCP over UMTS. In *Radiovetenskap och Kommunikation 02*, RVK, 2002.

[7] H. Elaarag. Improving TCP performance over mobile networks. *ACM Computing Surveys*, 34(3):357–374, 2002.

[8] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, 1993.

[9] C. P. Fu and S. C. Liew. TCP veno: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications*, 21(2):216–228, 2003.

[10] F. Gustafsson. *Adaptive Filtering and Change Detection*. Wiley, West Sussex, 2000.

[11] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review*, 18:314–329, 1988.

[12] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3), 2002.

[13] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[14] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of Operational Research Society*, 49:237–252, 1998.

[15] F. Khan, S. Kumar, K. Medepalli, and S. Nanda. TCP performance over CDMA2000 RLP. In *Proc. IEEE 51st VTC'2000-Spring*, pages 41–45, 2000.

[16] S. Liu, T. Basar, and R. Srikant. Controlling the Internet: A survey and some new results. In *Proc. 42nd IEEE Conference on Decision and Control*, pages 3048–3057, Maui, Hawaii USA, 2003.

[17] S. H. Low. A duality model of tcp and queue management algorithms. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, 2000.

[18] S. H. Low, L. Peterson, and L. Wang. Understanding Vegas: a duality model. *Journal of ACM*, 49(2):207–235, 2002.

[19] S. H. Low and R. Srikant. A mathematical framework for designing a low-loss, low-delay Internet. *Networks and Spatial Economics*, January-February 2003. special issue on "Crossovers between Transportation Planning and Tellecommunications".

[20] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: bandwidth estimation for enhanced transport over wireless links. In *MobiCom*, Rome, Italy, 2001.

[21] N. Möller and K. H. Johansson. Influence of power control and link-level retransmissions on wireless TCP. In *Quality of Future Internet Services*, volume 2811 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[22] R. G. Mukthar, S. V. Hanly, and L. L. H. Andrew. Efficient Internet traffic delivery over wireless networks. *IEEE Communications Magazine*, pages 46–53, December 2003.

[23] F. Paganini, Z. Wang, S. H. Low, and J. C. Doyle. A new TCP/AQM for stability and performance in fast networks. In *Proceedings of IEEE Infocom 2003*, 2003.

[24] J. P. Pan, J. W. Mark, and S. X. Shen. TCP performance and behaviors with local retransmissions. *Journal of supercomputing*, 23(3):225–244, 2002.

[25] N. K. G. Samaraweera. Non-congestion packet loss detection for TCP error recovery using wireless links. *IEE Proceedings-Communications*, 146(4):222–230, 1999.

[26] P. Sarolahti, M. Kojo, and K. Raatikainen. F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts. *ACM SIGCOMM Computer Communication Review*, 33(2), 2003.