

# 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@s3.kth.se](mailto:krister.jacobsson@s3.kth.se)

February 14, 2004

## Abstract

Recent research has indicated that knowledge of Round Trip Time (RTT) and available bandwidth is crucial for efficient network control. In this contribution we discuss the problem of estimating these quantities. Based on a simple aggregated model of the network, an algorithm combining a Kalman filter and a change detection algorithm (CUSUM) 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, especially in scenarios where new cross-traffic flows cause bottleneck queues to rapidly build up which in turn induces rapid changes of the RTT.

We also analyze how wireless links affect the RTT distribution. It is well known that link re-transmissions induces 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 is proposed to counter this problem. Carefully selected (artificial) delays are added to packets re-transmitted on the link which makes the delay-distribution TCP-friendly. The information required for this algorithm is readily available at the link and consists of the actual delay-distribution induced by the link. The added delays are obtained from a non-convex program which due to its low complexity is easy to solve.

## 1 Introduction

Congestion control has been one of the key factors for the ability of TCP to handle the dramatic growth of the Internet. The original idea, introduced in [14], was to adjust the transmission rate

based on the loss probability. The original implementation TCP Tahoe of this mechanism was further refined into what is known as TCP Reno and this algorithm (together with some of its siblings) is now the dominating transport protocol over the world.

TCP Reno is window-based which means that each sender has a window that determines how many packets in flight are allowed at any given time. The regulation is achieved by adjusting this window. For a network path with available (or fair) bandwidth  $b$  and Round-Trip-Time  $RTT$ , the optimal window size is

$$W = b \times RTT. \quad (1)$$

If all users employed this window, there would be no queues and the link capacities would be fully utilized, in other words maximum throughput would be achieved. Using loss probability 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 overflow (causing loss of throughput since packets have to be re-transmitted).

Several methods to cope with these shortcomings have been suggested. In TCP Vegas [5] 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 3). One interpretation [23] of this algorithm is that it estimates the round trip queuing delay and sets the rate proportional to the ratio of its round trip propagation delay to the queuing delay. Both round trip estimates are obtained from measurements of the RTT.

The links can also be used to indicate congestion in a more sophisticated manner than just dropping packets. In random early detection (RED) [9], packets are dropped even before the buffer is full with a probability that increases with the queue length. RED can thus be seen as a way of indirectly signalling the queue length to the source. In explicit congestion notification (ECN) the links add information concerning their status to the packets forwarded. As there is only one bit available in the packet header for ECN, clever coding is required. REM (Random Exponential Marking) [27] is one example of this.

In a remarkable contribution [18, 17] it was shown that the rate control problem can be solved in a completely decentralized manner. Each source has a (concave) utility function of its rate and the 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 and 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 and a number of contributions in this direction have been made. Algorithms can be classified as: 1) Primal, where the control at the source is dynamic whereas the link uses a static law. 2) Dual, where the link uses a dynamic law and the source control is static. 3) Primal-dual, where dynamic controls are used both at the source and links. We refer to [20, 21] for nice overviews of these methods. By appropriate choice of utility function even other non-optimization based protocols, such as TCP Reno, can be interpreted as distributed algorithms trying to maximize the total utility [22, 21]. TCP Vegas can be classified as a primal-dual algorithm as it uses queueing delay as link price which is a dynamic variable. The rate  $x_i$  for source  $i$  is adjusted according to

$$\dot{x}_i = \frac{1}{(d_i + q_i(t))^2} \operatorname{sgn} \left( 1 - \frac{x_i(t)q_i(t)}{\alpha_i d_i} \right)$$

where  $d_i$  is the round trip propagation delay, where  $q_i(t)$  is the total queueing delay on path  $i$  at time  $t$  and where  $\alpha_i$  is the set-point for the number of packets queued along the path. It is shown in [22,

21] that this corresponds to the utility function

$$\alpha_i d_i \log x_i$$

for which it has been shown [17] that proportional fairness is achieved.

It is pointed out in [21] that TCP Vegas seems better suited for high speed networks compared to Reno combined with RED which becomes unstable at high speeds [12]. In fact, it is by now generally recognized that network quantities such as queueing delay and round-trip-time are essential for efficient congestion control when ECN is not used, c.f., e.g., TCP Vegas and the algorithm in [28]. This can be given a simple intuitive interpretation: The ideal window is (1). Hence, the more accurate estimates of the RTT and the available bandwidth that are available, the closer to this 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 a delay increase. The queues can be seen as a way to smooth out the uncertainty in the RTT and bandwidth estimates. It is also clear that higher throughput than what Reno can achieve is possible as this algorithm operates at the other end of the spectrum filling up the bottle-neck completely before reacting.

One of the focal points of this contribution is to take a closer look of short range RTT from a statistical perspective. There are many studies of the statistics of network quantities such as byte sizes and RTT, see e.g. [3, 4]. There is also work on estimation of RTT at the link level [16]. However, very few studies have considered RTT estimation from the perspective of TCP itself. The raw RTT measurements, obtained from received packet acknowledgements, also include delays caused transient effects in the network that can be 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 and should hence be filtered out. In present versions of TCP (Reno, SACK, etc), this is done with a first order low-pass filter. However, the underlying RTT may suddenly change due to the appearance of a long lived cross-flow somewhere along the path. It is important, c.f. the discussion above, for the controller to react quickly to such

changes as otherwise buffers will start to build up unnecessarily with greater risk of packet loss and increased delay as consequences. This is not possible with a first order filter. In Section 2 we discuss alternative approaches to this problem.

Standard TCP protocols encounter some difficulties in heterogeneous, networks, e.g. when wired and wireless links are mixed. In these circumstances as the packet loss and packet delays on wireless links may be interpreted as congestion by the TCP. There are different approaches to counter act this problem. TCP Westwood can be seen as a way of modifying the transport protocol itself to improve the throughput. Another such protocol is presented in [31]. There are also other methods that try to differentiate loss as being due to congestion or lossy wireless transmissions, have also been proposed [6, 30, 10]. Performance enhancing proxies (PEP) is an alternative that has been proposed at link level. Here, split connection schemes (which introduces a virtual user at the link which acts as receiver to the source and source to the receiver) and interception schemes have been proposed. In the latter approach, passing acknowledgements are monitored and dropped if they indicate packet loss due to link-level retransmissions. A third alternative is to let the receiver control the transmission via its advertised window. We refer to [8, 26] for further details.

In Section 3 we study the case when there lost packets are retransmitted locally in a set-up that closely resembles 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 and that this triggers spurious time-outs which reduce throughput. We also propose a simple solution to this problem where the delay distribution is shaped in the link itself, before packets are retransmitted. Another contribution in this direction is [29].

## 2 Round-Trip Time Estimation

A packet switching communication network is a highly nonlinear complex dynamical system that

severe serious time delays and with unclear optimal objective (network throughput? user throughput? response time?). In this part of the paper we argue why online network state estimation is important and continues this discussion with presenting some initial results under this subject.

The Round Trip Time is the required time for a packet to travel both ways to the receiving end and back. In a network it is the sum of the aggregated link delay,  $d_l$ , propagation delay,  $d_p$ , and queue delay,  $d_q$ , that a packet experience when it travels through the network. Let  $\tau$  denote RTT and assume that the packet needs to pass  $m$  nodes, with corresponding downstream link, then

$$\tau(t) = \sum_{i=1}^m (d_{l,i} + d_{p,i} + d_{q,i}) \quad (2)$$

where  $t$  is the time instant the packet is received. Assume, for now, that the path through the network remains constant during a session. The propagation and link delay only contributes with a bias and all the major RTT dynamics is purely dependent on how the queues along the paths evolve.

The queue length at node  $i$ ,  $q_i(t)$ , is directly dependent on the traffic arrival process (arrival rate and packet size distribution scaled by the related link capacity  $c_i$ ). As illustrated in Fig. 1, the phys-

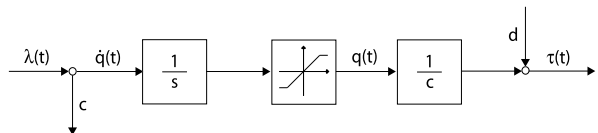


Figure 1: Node contribution to RTT ( $\lambda(t)$  traffic arrival rate).

ical queue dynamics is simply a saturated integrator. Denote the time instant for the concerned packet arrival at node  $i$  with  $t_i$  ( $t_1$  is then the moment the packet is sent and  $t = t_m + \tau_m$ ). Then, as queues evolve during the packet propagation through the network, the total RTT at time  $t$  becomes

$$\tau(t) = \sum_{i=1}^m \tau_i(t_i) = \sum_{i=1}^m \frac{q_i(t_i)}{c_i} + \text{bias} \quad (3)$$

To be able to perfectly predict the RTT fluctuations, we have to predict the traffic arrival rate

for every node along the path with no prior data about the network, and with only implicit information from a single delayed measurement.

## 2.1 RTT as a network metric

The actual RTT measurement is accurate. Due to the burstiness of the traffic in the Internet it is very fluctuating in its characteristic; too noisy to be used as raw data in congestion control applications. Since network queues build up and decrease very rapidly when additional traffic arrives the RTT time constant is only fractions of its value. This means that a single RTT sample is useless in the sense that it does not provide much information about what RTT next sent packet will experience the moment it is sent.

The goal in network congestion control is to reach a throughput equal to the available bandwidth. Both throughput and available bandwidth are not momentary quantities but defined as averages over some time frame. Instead of considering the instantaneous RTT a mean value would be a better metric. In other words we want to low-pass filter the true RTT and use the low frequency component. In TCP a first order exponential filter is used to get a suitable RTT estimate that is used in the Retransmission Timeout (RTO) computation [15]

However RTT have high frequency characteristics that is desirable to detect. To be able to follow step changes in the RTT mean value due to phenomena as increased network load, additional competing traffic or sudden path changes an adaptive filter with change detection could be more appropriate.

## 2.2 Modeling and change detection

By regarding the “true” RTT as a smooth signal and the high frequency component of it as noise disturbance, we can model the signal as a standard example of noisy observations of a constant that is exposed to step changes in the mean:

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

The noisy characteristic of RTT is modeled as the measurement noise  $e_t$  with variance  $R_e$ , the step

changes as the process noise  $v_t$  with variance  $R_v$ , and the discrete parameter  $\delta_t$  is 1 if a change occurs at time  $t$  and 0 otherwise. Estimating the sequence  $\delta^N = \{\delta_0, \delta_1, \dots, \delta_N\}$  is a so called *segmentation* problem.

Under the assumption of Gaussian noise components the Kalman filter is the optimal linear estimator [2]. Whenever the used surveillance filter detects a mean change,  $\delta_t$  is set to one and the process noise is triggered which makes the Kalman filter more sensitive.

The principle of change detection is illustrated in Fig. 2. A *distance measure* as e.g. the residuals are

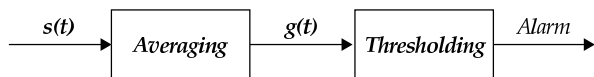


Figure 2: The principle of change detection.

averaged to get a test statistic that is then thresholded. If an alarm is given suitable action is taken and the test statistic is reset.

In our experiments we have chosen to use a one sided CUSUM [1] test to estimate the sequence  $\delta^N$ . Combining the problem specific Kalman estimator with the CUSUM test yields the following adaptive filter

The CUSUM KALMAN (CK-) filter:

$$\begin{aligned} \hat{x}_t &= \hat{x}_{t-1} + K_t(y_t - \hat{x}_{t-1}) \\ K_t &= \frac{P_{t-1}}{P_{t-1} + R_e} \\ P_t &= (1 - K_t)P_{t-1} + R_v \\ \epsilon_t &= y_t - \hat{x}_{t-1} \\ g_t &= \max(g_{t-1} + \epsilon - \xi, 0) \\ \delta_t &= 1 \Leftrightarrow R_v \neq 0, \text{ and alarm if } g_t > h \end{aligned}$$

After an alarm, reset  $g_t = 0$ . Else  $\delta_t = 0 \Leftrightarrow R_v = 0$ .

**Design parameters:**  $\xi, h$ .

**Output:**  $\hat{x}_t$ .

The CK-filter have two degrees of freedom, the negative drift  $\xi$  and the alarm threshold  $h$ . These parameters are tuned to adjust the sensitivity in the detection procedure, this have been done in line with the recommendations in [1]. The same values have been used in all our initial experiments with satisfying results, and the filter seems robust to parameter changes.

By manipulating the values of the variances we can influence the filter behaviour. Instead of triggering the process noise for only one sample it could e.g. be chosen to decay during a certain number of samples when an alarm is given. In the examples described later on both the process noise and measurement noise variance is set equal to the variance of the RTT samples.

### 2.3 Data collection

The Round Trip Time is continuously monitored in many communication networks. Statistical data is collected to analyze topography, network load, user behaviour etc. Most of the work is with respect to the longer timescale (hours, days, weeks). But for congestion control applications the RTT time granularity is in fractions of seconds (typically  $\approx 10 - 500$  ms).

The objective was to capture the rapid changes that the RTT undergoes when queues build up when e.g. the traffic load increases abruptly. A scenario like that would probably trigger the RTO mechanism in TCP which means lack of samples during the transients. 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.

Instead of monitoring the Internet for TCP RTT time series the ping tool was modified to fulfill our needs. With the new tool, *ping-time*, the sending interval can be assigned freely as long as the limitations of the cpu are not saturated. By sending a dense stream of small packets the RTT is monitored effectively without affecting the network too much.

### 2.4 Testing the CUSUM KALMAN filter

To get data to test the CK-filter on, we measured the RTT between our university office at KTH and the CAIDA web site. A path of normally 20 hops, including the Atlantic link, and with a mean RTT of approximately 190 ms. The ping-time tool was used with the sending interval set to 30 ms. According to our measurements the path is normally uncongested with the result that the RTT almost shows a deterministic behaviour with a low variance. However sporadically the traffic load in-

creased with the result of suddenly increased and fluctuating queues that propagates to the RTT.

The proposed filter was used on the collected data sets. The measurement noise variance,  $R_e$ , was set to the variance of the RTT samples and whenever the detection filter detected a mean change  $R_v$  was set equal to  $R_e$ . The design parameter of the CUSUM test was experimentally set to  $\xi = 0.005$  and  $h = 0.05$  in all experiments.

A detection of a sudden step in the RTT mean value can be seen in Fig. 3. The upper plot shows the sampled RTT values together with curves where the data is processed with the Kalman detection filter and, as a reference, the TCP first order exponential filter [15] 'gain' set to 0.9. In the lower plot the test statistic  $g_t$  is plotted and the alarms are encircled. The CK-estimate stays very smooth but still manage to detect a mean change of only 7%. The exponential filter is more sensitive to noise

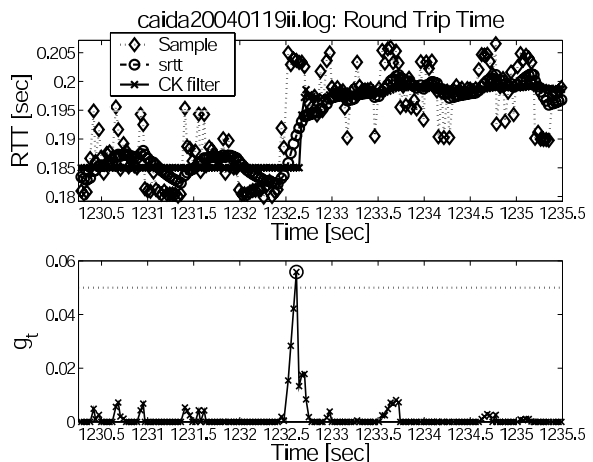


Figure 3: Static change in RTT mean value.

and periodic fluctuations. As the change in mean is moderate it adapts quite fast in this example. 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 an explanation. However a re-routing inside the network is also a possibility.

In Fig. 4 we see a large change in the mean of about 100%. Even if the probing packets are sent with only 30 ms between we do not capture the queue building up. The result becomes a step in the RTT. In this scenario the CK-filter reacts after

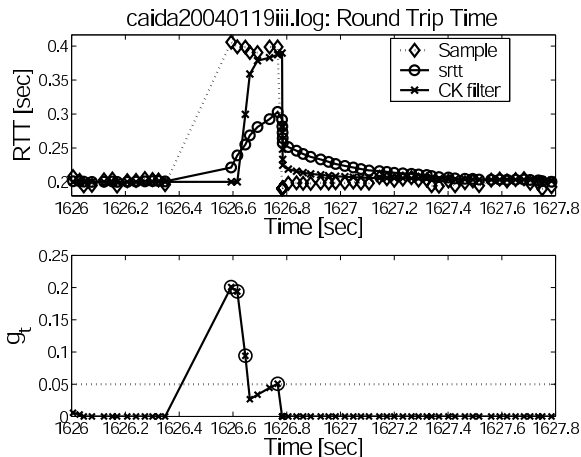


Figure 4: Sudden change in RTT mean value.

a few samples, actually 3, and is much faster than the exponential filter. However, this time it was no static mean change, the queue vanishes after half a second and the decreases down to the original level. The CK-filter reacts on that too and adapts almost immediately when the exponential filter is lagging.

In a real protocol application the sampling time is typically larger and the measurements tend to be more spiky. The probability of hitting a plateau as in Fig. 4 with more than a few samples is low. By filtering out such events the RTT estimate is kept at the appropriate level.

The two given examples shows the main feature with the CUSUM KALMAN filter; smooth estimates but still fast adaption when drastic changes occur.

### 3 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.

#### 3.1 System overview

When using TCP over a wireless link, there are several interacting control systems stacked on top of each other, illustrated in Figure 5. 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 modelling the lower layers, we can investigate effects the link layer control has on TCP performance. We refer to our previous paper [25] for further details on the radio model.

#### 3.2 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 [7], which is what we will use.

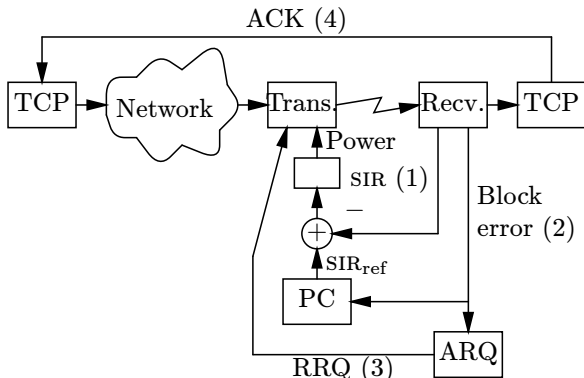


Figure 5: System overview, including four feedback loops marked (1) – (4).

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  $\Delta$ . It decreases  $SIR_{ref}$  by  $\Delta$  for each successfully received block, and increases  $SIR_{ref}$  by  $9\Delta$  each time a block is damaged.

The process can be modelled as a discrete Markov chain, where state  $k$  corresponds to  $SIR_{ref} = k\Delta$ . 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  $SIR_{ref} = r$ . 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  $\Delta$ , it is straight forward to compute the stationary distribution of the Markov chain. Figure 6 shows the stationary distribution for an example BPSK channel (see [25] for the parameters) and three different values for  $\Delta$ .

### 3.3 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 con-

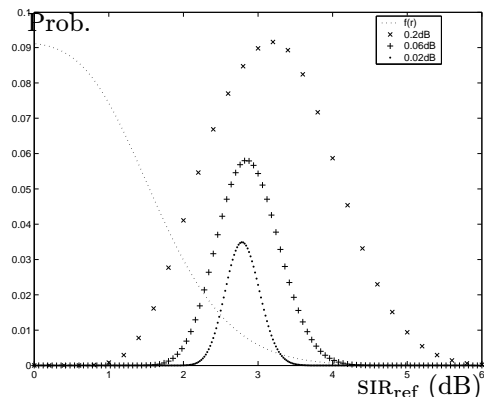


Figure 6: Stationary distribution for the power control. Each mark represents one state of the power control, the corresponding value of  $SIR_{ref}$ , and its stationary probability. The dotted curve is the threshold-shaped function  $f(r)$ , scaled to fit in the figure, which represents the block error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

trol), and damaged blocks are scheduled for retransmission. We will consider one simple retransmission scheme, the (1,1,1,1,1)-Negative Acknowledgement scheme [19], 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.

### 3.4 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. We let  $n$  denote the number of radio blocks needed for the packet size of interest. For the links we consider, we have  $n \leq 10$ .

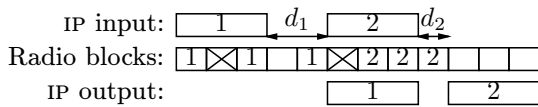


Figure 7: IP packets divided into radio blocks

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 re-assembled 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  $\Delta = 0.06\text{dB}$ , and  $n = 2$ , is shown in Table 1.

|     |      |     |     |     |     |      |      |
|-----|------|-----|-----|-----|-----|------|------|
| $d$ | 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.

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 acknowledgement, is delayed too long. Let  $\text{RTT}_k$  denote the roundtrip time experienced by packet  $k$  and its corresponding acknowledgement. The TCP algorithms estimates the mean and deviation of the roundtrip time. Let  $\widehat{\text{RTT}}_k$  and  $\hat{\sigma}_k$  denote the estimated roundtrip time and deviation, based on measurements up to  $\text{RTT}_k$ . 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

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

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 estimation is perfect, and that the timeout value is set to  $R_{\text{TO}} = E(\text{RTT}) + 4\sigma(\text{RTT})$ . 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  $n, \Delta$ , we typically get a probability of spurious timeout on the order of 0.5–1% [25].

### 3.5 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

$$P_{\text{TO}}(X) = P(X > E(X) + 4\sigma(X)) \quad (5)$$

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  $P_{\text{TO}}(X)$  is invariant under the addition of constant delays.

For distributions we know are friendly to TCP,  $P_{\text{TO}}$  is small. For a general distribution, assuming

only independence and finite first and second moments,  $P_{\text{TO}}$  is bounded by Chebyshev's inequality. Comparing these values,

$$X \text{ uniform} \implies P_{\text{TO}}(X) = 0 \quad (6)$$

$$X \text{ normal} \implies P_{\text{TO}}(X) \approx 6.3 \cdot 10^{-4} \quad (7)$$

$$X \text{ wireless} \implies P_{\text{TO}}(X) \sim 100 \cdot 10^{-4} \quad (8)$$

$$X \text{ arbitrary} \implies P_{\text{TO}}(X) = 625 \cdot 10^{-4} \quad (9)$$

we see that the two friendly distributions yield a  $P_{\text{TO}}$  at least two orders of magnitude below the worst case given by Chebyshev, while the wireless delay yields a significantly higher  $P_{\text{TO}}$ , 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 mechanism 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.

### 3.6 Introducing additional delays

Assume that we have a discrete delay distribution for  $X$ ,  $P(X = d_i) = p_i$ , where  $d_i < d_{i+1}$ . It is typical, but not required, that also  $p_i \geq p_{i+1}$ . Let  $\mu$  and  $\sigma^2$  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  $\tilde{X}$ ,  $P(\tilde{X} = d_i + x_i) = p_i$  (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,  $\epsilon$ , for  $P_{\text{TO}}(\tilde{X})$ , and minimize  $E(\tilde{X})$  under the constraint that  $P_{\text{TO}}(\tilde{X}) \leq \epsilon$ . 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  $P_{\text{TO}}(\tilde{X})$  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  $\delta$  is a robustness margin. We impose the additional constraints  $P_{\text{TO}}(\tilde{X}) = c$ ,  $x_i + d_i \leq d_{k+1}$  for  $i \leq k$ , and  $x_i = 0$  for  $i > k$ . Then, for any  $x_i$  satisfying these new constraints, we will have  $P_{\text{TO}}(x) = \sum_{i \geq k+2} p_i \leq \epsilon$ . Written as an optimization problem, we have

$$\min E(\tilde{X}) \quad (10)$$

$$P_{\text{TO}}(\tilde{X}) = c \quad (11)$$

$$x \geq 0 \quad (12)$$

$$x_i \leq d_{k+1} - d_i \quad (13)$$

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

$$\min p^T x \quad (14)$$

$$x^T S x + b^T x + \alpha = 0 \quad (15)$$

$$0 \leq x \leq m \quad (16)$$

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  $x = (0, \dots, 0, x_j, m_{j+1}, \dots, m_k)^T$ . When the optimum

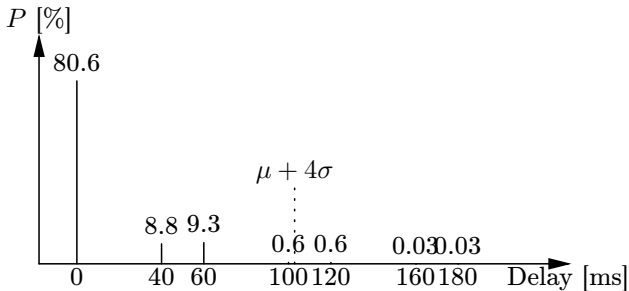


Figure 8: Original delay distribution

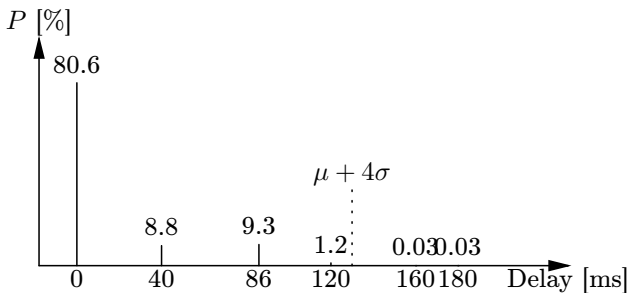


Figure 9: Optimized delay distribution

has this form, it means that the cheapest way to to increase  $P_{\text{TO}}$ , 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  $E(X) \approx 10.6$  ms, and  $P_{\text{TO}}(X) \approx 0.68$  %.

With  $\epsilon = 0.1\%$  and  $\delta = 10$  ms, the above optimization procedure yields  $k = 4$  and the optimal additional 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  $P_{\text{TO}} < \epsilon$ , what we actually get is  $P_{\text{TO}} \approx 0.06\%$ .

### 3.7 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  $P_{\text{TO}}(\tilde{X} + V)$ . Since  $P_{\text{TO}}$  is invariant under the addition of constant delays, we can make the convenient assumption that  $E(V) = 0$ .

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

$$\begin{aligned} P_{\text{TO}}(\tilde{X} + V) &= P(\tilde{X} + V > c') \\ &= \sum_i p_i P(V > c' - x_i - d_i) \\ &\leq \sum_{i=1}^k p_i P(V > \delta) + \sum_{i>k} p_i \\ &\leq P(V > \delta) + \epsilon \quad (17) \end{aligned}$$

This bound shows that if the delay variations in the rest of the network are small enough relative to our robustness parameter  $\delta$ ,  $P_{\text{TO}}(\tilde{X} + V)$  will not be much larger than  $\epsilon$ . 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  $\delta$ . A more precise bound can be calculated using additional information about  $p_i$  and the distribution of  $V$ .

### 3.8 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

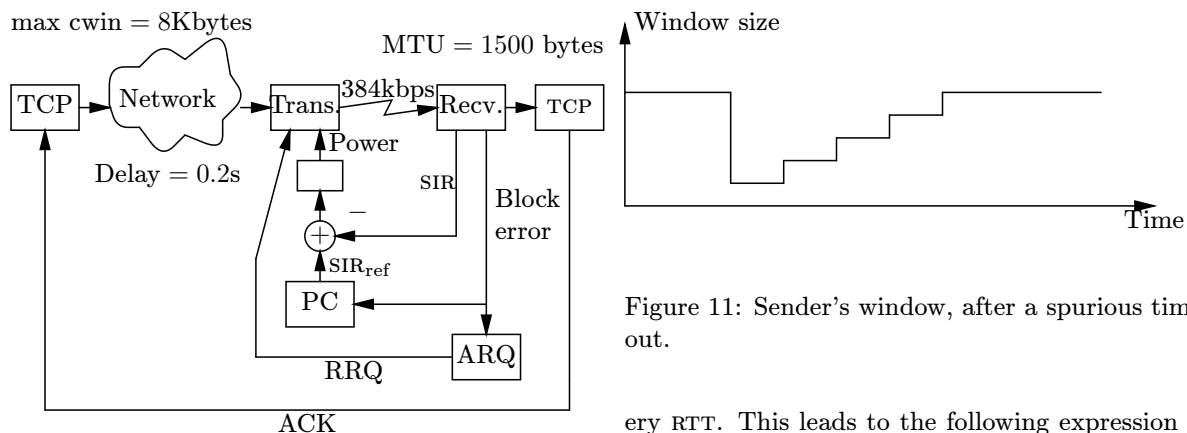


Figure 10: Numerical example

performance evaluation [19]: 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 roundtrip delay time, excluding the radio link itself, of 0.2 s.

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 roundtrip 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 ev-

Figure 11: Sender's window, after a spurious timeout.

ery RTT. This leads to the following expression (a more general formula is derived in [25]).

$$\text{Throughput} = \frac{w}{E(\text{RTT})(1 + 10P_{\text{TO}})} \quad (18)$$

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  $P_{\text{TO}}$ . 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] *Adaptive Filtering and Change Detection*. Wiley, West Sussex, 2000.
- [2] *Linear Estimation*. Information and System Sciences. Prentice Hall, New Jersey, 2000.
- [3] M. Allman. A web server's view of the transport layer. *ACM SIGCOMM Computer Communication Review*, 30(5), 2000.

- [4] M. Allman and V. Paxson. On estimating end-to-end network path properties. *ACM SIGCOMM Computer Communication Review*, 31(2), 2001.
- [5] 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.
- [6] 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.
- [7] A. Dahlén and P. Ernström. TCP over UMTS. In *Radiovetenskap och Kommunikation 02*, RVK, 2002.
- [8] H. Elaarag. Improving TCP performance over mobile networks. *ACM Computing Surveys*, 34(3):357–374, 2002.
- [9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, 1993.
- [10] 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.
- [11] M. Gerla, M.Y. Sandidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo. TCP westwood: Congestion window control using bandwidth estimation. In *IEEE Globecom'01*, San Antonio, Texas, November 2001.
- [12] C. Holot, V. Mishra, D. Towsley, and W.-B. Gong. A control theoretic analysis of red. In *Proceedings of IEEE Infocom 2001*, 2001.
- [13] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21(6):879–894, 2003.
- [14] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review*, 18:314–329, 1988.
- [15] V. Jacobson. Congestion avoidance and control. In *Proc. of SIGCOMM*, volume 18.4, pages 314–329, 1988.
- [16] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3), 2002.
- [17] 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.
- [18] F.P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.
- [19] 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.
- [20] 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.
- [21] 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 Telecommunications".
- [22] 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.
- [23] S.H. Low, L. Peterson, and L. Wang. Understanding Vegas: a duality model. *Journal of ACM*, 49(2):207–235, 2002.
- [24] B. Melander, M. Björkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *IEEE Globecom'00*, San Francisco, CA, November 2000.

- [25] N. Möller and K.H. Johansson. *Quality of Future Internet Services*, volume 2811 of *Lecture Notes in Computer Science*, chapter Influence of power control and link-level retransmissions on wirelessTCP. Springer-Verlag, 2003.
- [26] R.G. Mukthar, S.V. Hanly, and L.L.H. Andrew. Efficient Internet traffic delivery over wireless networks. *IEEE Communications Magazine*, 2003.
- [27] S. Athuraliya nad V.H. Li, S.H. Low, and Q. Yin. Rem: active queue management. *IEEE Networking*, 15(3):48–53, 2001.
- [28] 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.
- [29] 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.
- [30] N.K.G. Samaraweera. Non-congestion packet loss detection for TCP error recovery using wireless links. *IEE Proceedings-Communications*, 146(4):222–230, 1999.
- [31] 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.