ATNP/WG2

WP .....

20 Sept 1996

AERONAUTICAL TELECOMMUNICATIONS NETWORK PANEL

Working Group 2

Alexandria, Virginia, USA

7 - 15 October 1996

On Adaptive Retransmission Timers in OSI TP4

Prepared by O. Rose

SUMMARY

This paper presents a simulation study that quantifies the benefits achievable with an adaptive (i.e. dynamically adjusted) retransmission timer applied in the OSI transport protocol class 4. A recommendation for the ATN SARPs is offered.

Contents

1 Problem Description

The OSI transport protocol class 4 (TP4) contains plenty of timers, most of which have a certain relationship to each other, or even to timers being used within other layers (such as the reassembly timer within CLNP). One of the most important timers, both regarding the performance achievable when transferring data, and with respect to the (avoidable) load put upon the underlying network layer, is the retransmission timer.

The sender will start this timer whenever it transmits a data packet, expecting the corresponding acknowledgement to arrive before the associated timer expires. If, however, the timer expires before the acknowledgement has been received, the sender assumes that the packet was lost, and will retransmit it.

The 'optimum' value of the retransmission timer thus is clearly related to the round-trip time (RTT) of a connection (i.e. the time between the transmission of a packet, and the reception of the associated acknowledgement). Since the load found within the underlying network will vary (i.e. queue sizes found within intermediate systems being traversed by a packet will vary), it becomes clear that the RTT will also vary to a certain degree.

All in all, the 'optimum' value of this timer should thus be

- as close to the RTT experienced as possible, however,
- be sufficiently large so as not to falsely indicate a packet loss, due to delays the packets (both data and ACK) experience when traveling through the network.

In other words, the 'optimum' value of the retransmission timer both depends upon the **actual path** used to reach the destination, and the **actual load** found within the network.

Deciding upon the fixed value to use as the 'optimum' retransmission timer value is thus quite difficult. Assume a sender talks to some destination nearby (possibly even within the same ATC). In that case, the retransmission timer can be (and in fact should be) quite small, e.g. in the range of several milliseconds. If, however, the sender talks to an airplane that is located far apart, the retransmission timer has to be set to a fairly large value (e.g. in the range of several seconds)[1]. To be prepared for both cases, as necessary if a fixed value is used, the one fixed value must be chosen large enough.

---

[1] *Otherwise, packets will falsely be recognized as being lost, since their retransmission timer expires. Assuming a retransmission timer value that is smaller than the RTT, each packet will at least be transmitted twice by the sender, since the sender will always deduce the packet has been lost, when its retransmission timer expires (given each packet has an associated retransmission timer). The total load found within the network will thus be twice as high as necessary to transmit a certain amount of data.*

This normally does not harm, as long as no packets are really lost. However, things become much worse if packets get lost. In that case, significant loss of throughput can be experienced, as will be shown in the next subsection using simulation.

1.1 Using manually configured retransmission timer values

The scenario used to visualize the problems of a fixed, manually configured retransmission timer is depicted in figure 1 below.

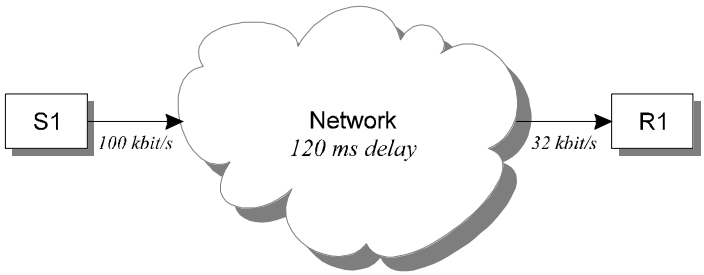

Figure 1: Scenario to demonstrate the benefit of adaptive retransmission timers

Therein, a sending application S_1 tries to transmit data over some network to a receiver R_1. Traversing the network will take approx. 120 ms, so the RTT is approximately 0.3 s (allow some time required to actually put each TPDU on the wire). However, to be save for a wide variety of different connections, the manually configured retransmission timer has been set to 5 seconds (e.g. to also be able to run over a connection that goes half way around the world, even to an aircraft currently flying there).

The receiver is connected via a 32 kbit/s link to the network. Due to the protocol overhead introduced by TP4 and CLNP, the data rate achievable between sender and receiver over this link is restricted to approx. 25 kbit/s. Data is transmitted using DT-TPDUs with a maximum size of 256 byte.

Finally, the network is simulated to be congested, so packets will get lost from time to time.

Figure 2 now depicts the stream of packets being received at R_1. Each dot in this figure represents the reception of a packet, with the abscissa indicating the time when the packet has been received, and the ordinate depicting the amount of data received (remember we consider a sender that tries to transmit a continuous stream of data to the receiver).

The most important thing that can be noted is the data rate achieved. According to figure 2, the sender is able to transmit only

$$\frac{30000 \cdot 8}{100} = 2400 \left[\frac{bit}{s} \right]$$

i.e. less than 10% of the maximum throughput achievable. Taking a closer look at the reception of packets over time to identify the reason why, it can be seen that there are large time spans

when almost no new data is received (e.g. from about t= 5 s to about t= 30 s). These time spans represent phases during which packet losses are recovered. Since 5-6 packets are lost in sequence, it takes about $5.5 \cdot 5$s to recover from this loss, i.e. 27.5 s are more or less uselessly spent with packet recovery activities.[2]

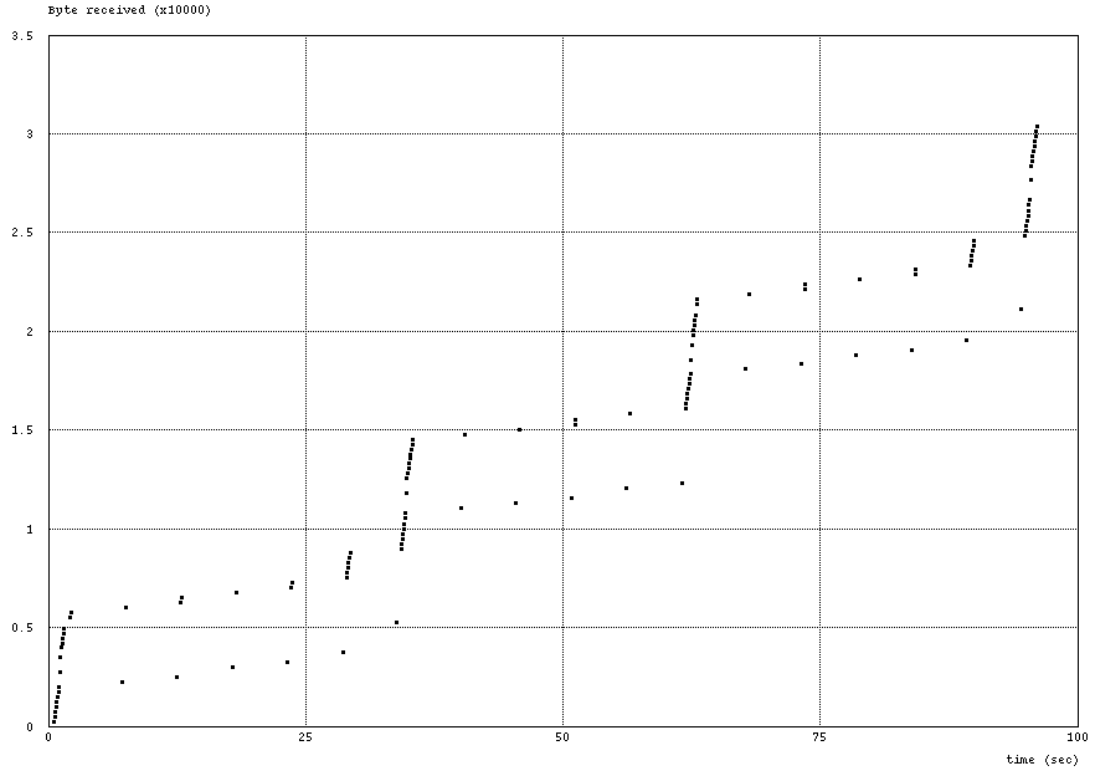

Figure 2: Stream of packets as seen at receiver, fixed retransmission timer

Within the scenario depicted above, however, the RTT really is only about 0.3 s, i.e. it should be possible to detect lost packets much quicker, and thus to reduce the time required to re-

---

[2]Note that this behaviour is a result of using a single retransmission timer for all outstanding packets, as currently implemented in the TP4 model of the European ATN simulation suite. This single timer will always supervise the oldest packet not yet acknowledged. If it expires, this packet (and only this packet) is retransmitted, and the timer is started again. Assuming the repeated packet makes its way to the receiver (as is the case for the simulation considered here), the next lost packet will not be repeated prior to timer expiration, leading to the large gaps between repeated packets, as found in the figure.
This situation could be improved by either using a separate timer for each packet, or by repeating upon timer expiration all packets currently not acknowledged. The first approach will however lead to the problems with wrongly set timer values described on page 1, and significantly increases implementation complexity due to the large number of timers that have to be administered. The second approach will lead to a burst of packets being generated whenever some packet has been detected to be lost, producing a high load for the network and increasing the risk of loosing some packet once again during recovery.

cover from packet loss. The only reason for the large RTT timer value was some safety consideration, since the (fixed) value selected during parameterization must have been chosen large enough to accommodate even the largest RTT occurring.

This can be avoided if, instead of using some fixed value, the retransmission timer value is dynamically adjusted to the characteristics of the current connection (i.e. its round-trip time). This approach shall be described in the next section.

2 Dynamic retransmission timers

Taking a look at the problems that a fixed retransmission timer value introduced (see section 1 above), it becomes clear that the actual value of the retransmission time should be derived from the round-trip time (RTT) experienced. Recall that the round-trip time denotes the time between the transmission of a packet, and the reception of the associated acknowledgement. Smaller round-trip times should thus lead to smaller values of the retransmission timer, and larger RTTs to larger values.

The RTT can easily be determined by the sending transport entity, if it remembers the time when a packet was transmitted. When the associated acknowledgement is received, the difference between the actual time and the time when the packet has been transmitted gives a measure of the RTT experienced. This observation forms the base for an adaptation of the retransmission timer.

2.1 Basic algorithm (TCP)

From the measured round-trip times, a running average value can be computed using an algorithm known as Exponential Aging. Application of this technique leads to the algorithm depicted below:

$$estimated\ RTT := estimated\ RTT + \alpha \cdot (measured\ RTT - estimated\ RTT) \qquad (1)$$

With α e.g. set to 1/8 (as proposed for TCP), this algorithm tracks the average round-trip time a TPDU experiences.

The retransmission timer itself must of course be set to a value higher than this average, to avoid triggering false packet retransmissions. To achieve this, the retransmission timer value (also called *retransmission timeout*, or *rto* for short) can, in the simplest case, be computed to

$$rto := \beta \cdot estimated\ RTT$$

with β e.g. set to 2 (as is the case for TCP). Figure 3 depicts the operation of such an adaptive timer[3]. Measured RTT values are indicated using small circles, with the running average computed according to equation (1) above shown in between those values. Finally, the *rto* derived is also shown, which is simply twice the value of the average.

Rem.: Within TP4, the retransmission timer is called „T1". Sometimes this name is used within the figures derived from simulation runs. However, it is always identical to the term „rto" used in the text.

---

[3]Note: The situation depicted here and in the next picture is used solely to demonstrate the operation of adaptive retransmission timers. It does not reflect the round-trip times that can be measured in the simple data transmission scenario used in chapter 1 to show reception of packets over time. Therein, round-trip times do much stronger correlate.

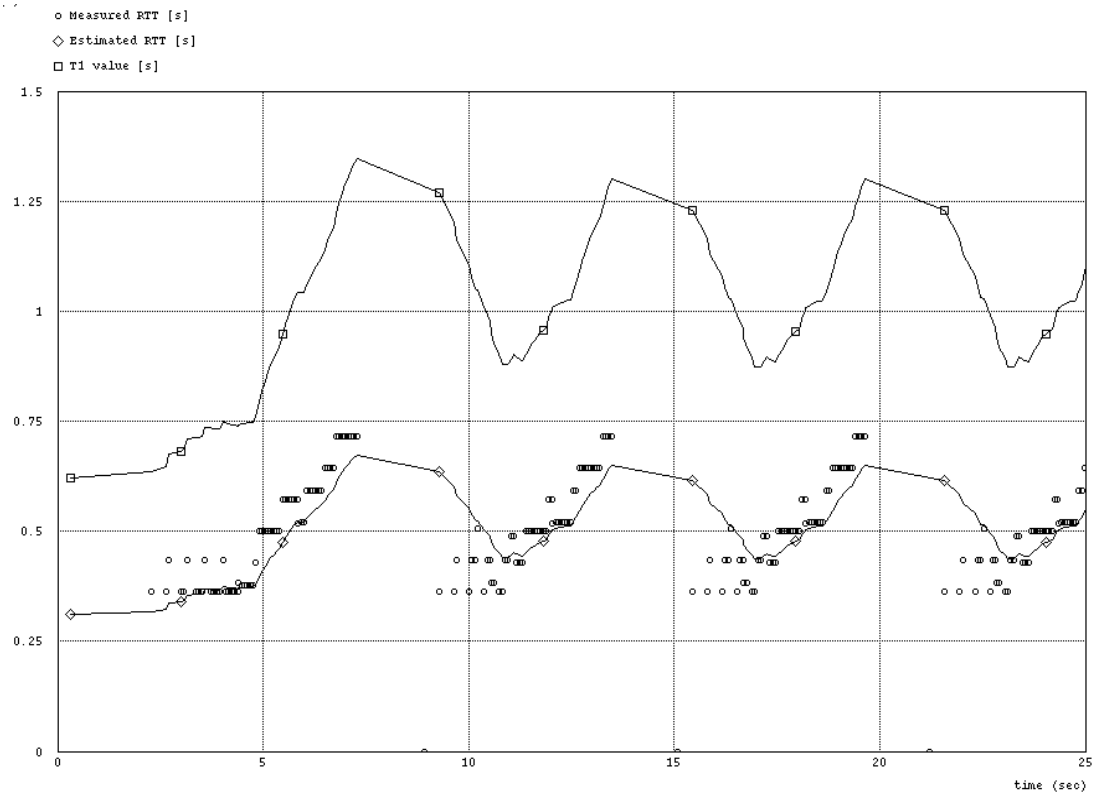

Figure 3: Tracking the average round-trip time using Exponential Aging

However, the question remains whether the constant factor β introduced above to derive the retransmission timer value from the averaged RTT is sufficient in all cases to avoid triggering of false alarms. One can imagine that, within a highly loaded network, the RTT will vary to a sufficient extent so that the algorithm falsely indicates a packet loss (i.e. the acknowledgement has been delayed to such an extent that the retransmission timer has expired). To prevent this from happening, one might e.g. also estimate the variance found within the RTTs measured, and compute the retransmission timer value from both the estimated average RTT, and its estimated variance. This is performed in an improved version of the algorithm, that has been introduced by Van Jacobsen in 1988 [1]. It will be described in the next sub-section.

2.2 Improved algorithm (Van Jacobsen)

To derive a stable operating retransmission timer (i.e. one that avoids false alarms to the greatest extend possible), it is better not to simply multiply the average RTT value with some constant. Van Jacobsen has proven this in [1], and proposed to adjust the *rto* value according to both the average round-trip time estimated, and the variance of this measure. Given a lightly loaded network, this variance will be quite small, leading to a *rto* value that is close to the average and thus allows quick recovery from lost packets. Given a highly loaded network, this variance (and thus the *rto* derived) might become quite large, so the constant factor would likely introduce false alarms, that can be avoided with the improved variant of the algorithm.

To facilitate computation of the retransmission timer value, instead of computing the variance, the mean deviation of the measured round-trip time is computed. This improved algorithm also primarily operates using the measured round-trip times. It is shown below:

$$
\begin{aligned}
error &:= measured\ RTT - estimated\ RTT \\
estimated\ RTT &:= estimated\ RTT + \alpha \cdot error \\
estimated\ deviation &:= estimated\ deviation + \gamma \cdot (|error| - estimated\ deviation) \\
rto &:= estimated\ RTT + 2 \cdot estimated\ deviation
\end{aligned}
$$

A value of 1/4 has been proposed by Van Jacobsen for parameter γ, with α kept at 1/8[4].

The operation of this improved algorithm when determining the value of the retransmission timer is depicted in figure 4. Note that the measured values are identical to those found in figure 3, only the computation of the retransmission timer value has been changed.

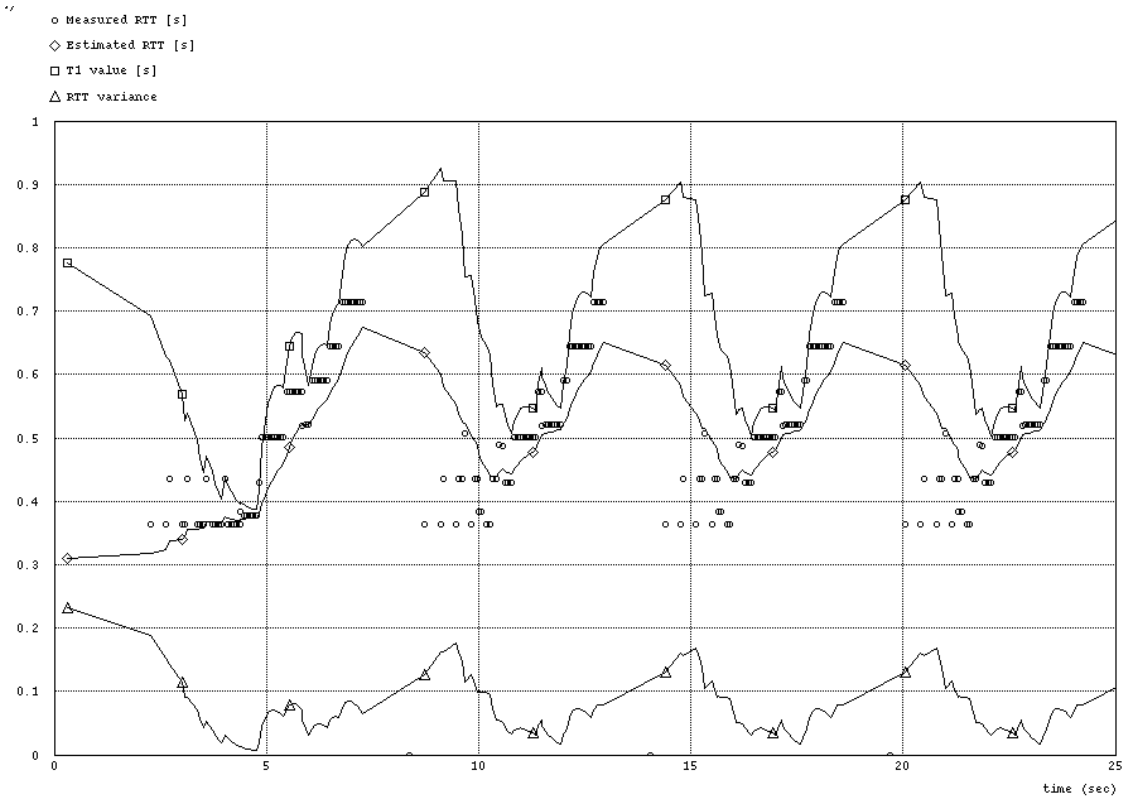

Figure 4: Improved computation of retransmission timer value

---

[4]Note that these values are primarily chosen in order to facilitate computation of the round-trip timer. Measured RTTs in real systems are often kept as integer values, indicating multiples of the timer granularity (which often is in the order of 100 ms). Updating the rto *thus can be done in a highly efficient manner using additions and shifts, avoiding multiplication of integers (or even worse floating point multiplication, if time values were kept as doubles).*

Again, the measured round-trip times that serve as the starting point for the algorithm are shown as small circles. The middle graph also depicts the running estimate for the average RTT.

The bottom graph shows the running estimate of the deviation of measured RTTs from the estimated average RTT. This estimate is used to determine how close the retransmission timer may be set to the estimated average RTT. The top graph finally displays the value computed for the retransmission timer.

As can be seen from the graph, the application of this algorithm will set the retransmission timer quite close to the estimated average RTT during times when the measured RTTs only mildly differ from the estimated average. On the other hand, if there are larger variations in the measured RTTs, the algorithm quickly reacts by increasing the value of the retransmission timer, thus avoiding false detection of packet losses due to delayed acknowledgements. Note further that computation of the retransmission timer introduces some asymmetry. The timer quickly goes up when measured round-trip times increase, but more slowly comes down afterwards. This behavior has also been suggested in [3] as a desirable feature to provide a stable operating retransmission timer.

To give an impression about the improvement in performance that can be achieved with the introduction of an adaptive retransmission timer, figure 5 below depicts a situation similar to figure 2, but now with an adaptive retransmission timer in use.[5]

[5] *Note that, despite a very similar appearance, scaling is quite different between figure 2 and figure 5. The latter indicates units of 100 kbyte, while the former indicates units of 10 kbyte, i.e. only 10% of the amount of data as depicted in figure 5.*

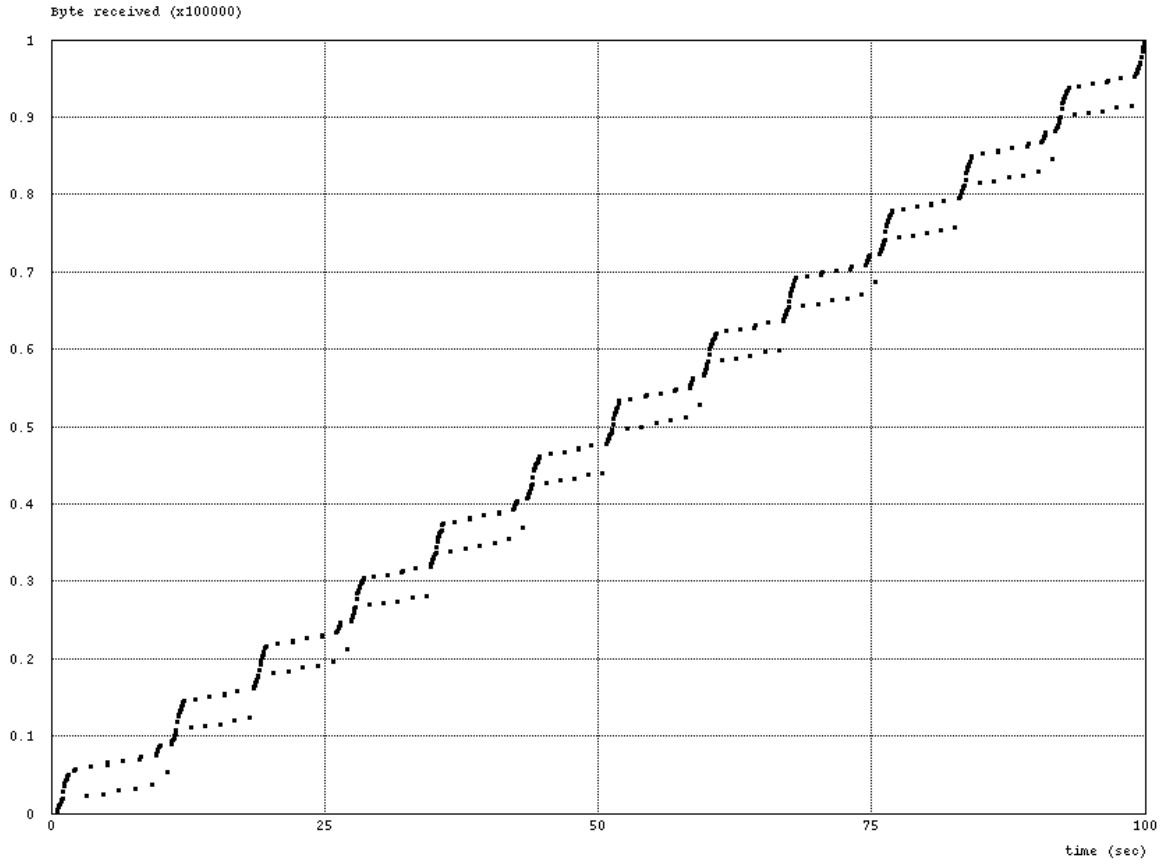

Figure 5: Stream of packets as seen at receiver, adaptive retransmission timer

Note that we still simulate a congested network, i.e. packets will get lost when they traverse the network. However, packet losses are now recovered much faster, since the adaptive retransmission timer now adjusts itself to a value that matches the underlying path (i.e. to a value of approximately 0.9 seconds).

The throughput achievable with both the fixed timer value and with an adaptive retransmission timer that is updated according to the algorithm proposed by Van Jacobsen is finally depicted below.

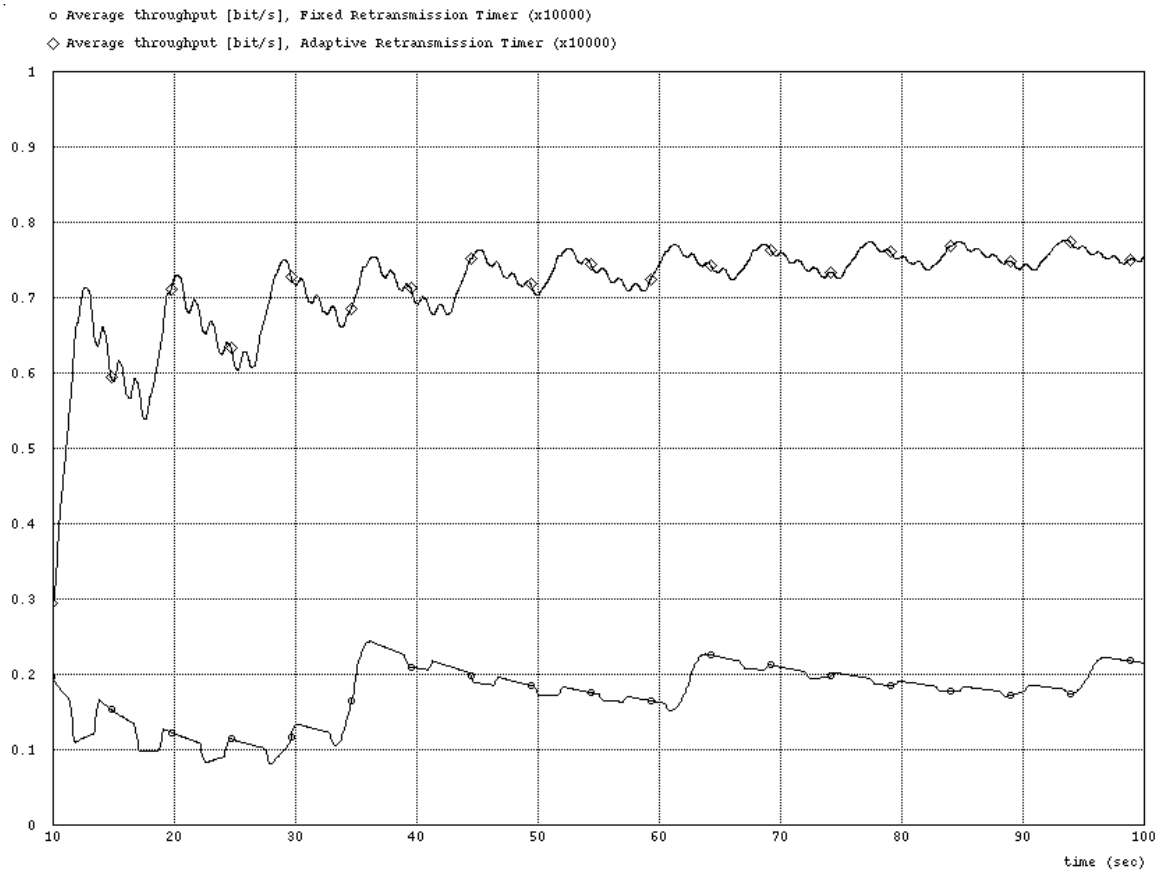

Figure 6: Throughput achievable with both the fixed and adaptive retransmission timer

Note that the throughput achievable with the fixed retransmission timer is limited to about 2 kbit/s, which is largely caused by a timer value that is too high for the actual connection simulated here.

Applying a retransmission timer that is able to adapt to the connection's characteristics, the throughput that can be achieved over the same congested network increases to a value of approximately 7.5 kbit/s.

2.3 Dealing with packet losses

The accuracy of the value derived for an adaptive retransmission timer, if computed using some variant of Exponential Aging, heavily depends upon the fact that the sequence of round-trip time samples used is an accurate measurement of the true round-trip time. This normally is the case, since we measure the time between the transmission of a data packet, and the reception of the associated acknowledgement.

Note, however, that this is not true if a packet has to be retransmitted! Assume packet p will get lost during transmission. After some time, the retransmission timer will expire, and the packet will get retransmitted. Somewhat later, the sender will receive an acknowledgement for

packet p. It is now not clear how the round-trip time shall be computed. Is it the difference between the actual time and the time the original packet has been transmitted, or shall the time of the last retransmission be used to compute the round-trip time.

As has been shown in a paper by Phil Karn and Craig Partridge [2], using neither of the two values is advisable. Using the time of transmission of the original packet might cause the round-trip time estimate to grow without bound when there is loss on the network. Using the time of the last retransmission is dangerous as well. In that case, the RTT estimate might stabilize at an unreasonably low value. Unnecessary data retransmissions do occur constantly, useful throughput drops sharply, and network bandwidth is wasted in that case.

Ignoring the samples derived from packets that have been retransmitted is thus the best advice. This approach works, provided that the true round-trip time never grows faster than the estimation algorithm can adapt. If, however, the round-trip time suddenly grows (e.g. due to using another path between source and destination), all samples will get discarded from that time on. In consequence, the retransmission timer value will never be updated again, but stick with a value that is too small for the actual connection. Again, unnecessary data retransmissions will occur, together with a very small throughput achievable and lots of network bandwidth wasted.

To make the algorithm robust against this case, also, some kind of exponential back-off should be used when a packet has to be retransmitted. In that case, the retransmission timer value is increased by a certain factor, so it will reach a value large enough to accommodate to the new path after a certain time. This algorithm, better known as Karn's algorithm in the TCP/IP community and described in [2], can be summarized with the following rule:

> *When an acknowledgement arrives for a packet that has been sent more than once, ignore any round-trip time measurement based on this packet. In addition, the backed-off retransmission timer value for this packet is kept for the next packet. Only when it (or a succeeding packet) is acknowledged without an intervening retransmission will the* rto *be recalculated from the sampled round-trip time.*

The behavior of the round-trip time estimation algorithm, when being applied to supervise the data transmission with packet losses example considered in the previous chapters, is depicted in figure 7 below.[6]

[6] *Note, however, that exponential back-off of the retransmission timer value in case of packet losses is not included in this figure, for the sake of clarity.*

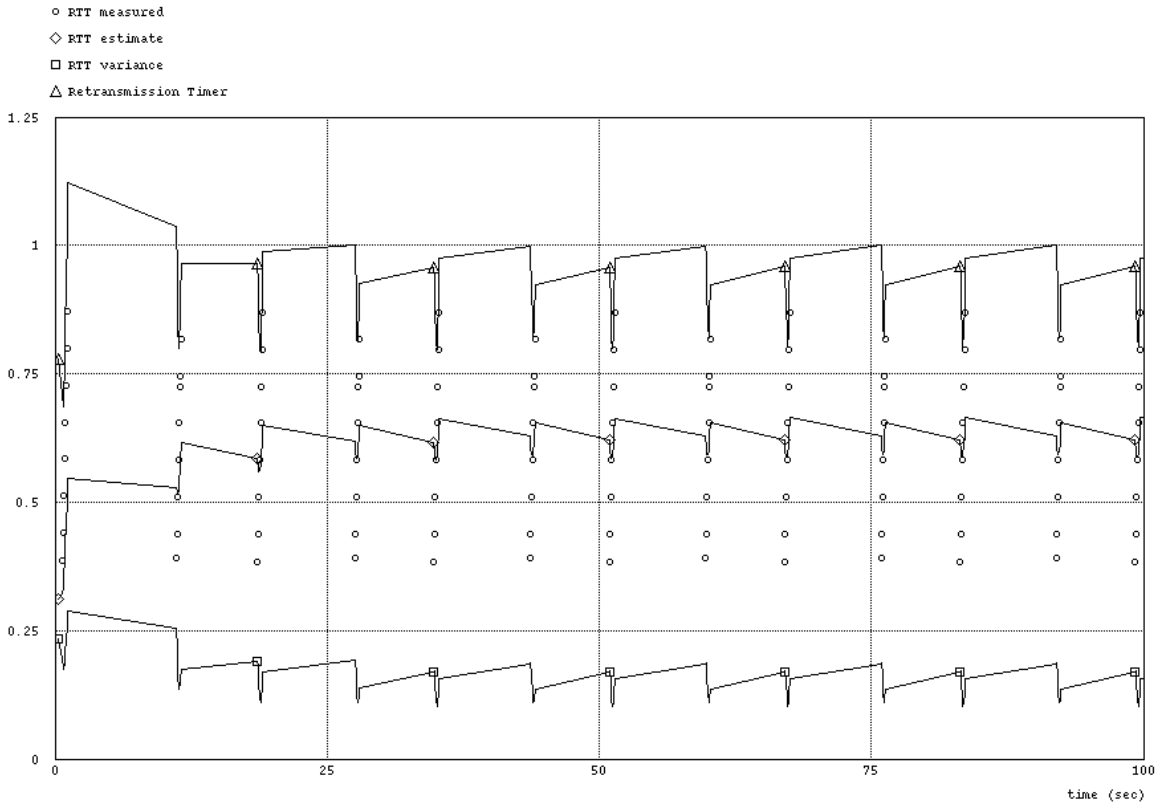

Figure 7: Operation of retransmission timer algorithm, according to Van Jacobsen and Karn/Partridge

As can be seen from the figure, only a very limited amount of samples (represented by small circles) are available to update the round-trip time estimate. The round-trip time of packets quickly increases, indicating that these packets are transmitted faster than the network can forward them, and finally will get discarded due to buffer overflow within the network. Afterwards, during the packet recovery phase, the sender will only see duplicate acknowledgements or acknowledgements for packets that have been retransmitted. However, these shall not be used to update the estimate of the round-trip time, as briefly described above. Only after recovery from packet losses has been finished, new samples will be available.

3 Conclusion

An adaptive retransmission timer within the OSI transport protocol class 4 (TP4) is basically useful for the following reasons:

- to improve the performance of data transmission, since more rapid packet recovery will reduce the average delay a packet experiences when traveling through the network.
- to reduce the amount of work required to (manually) parameterize the ATN stack;
- to reduce the risk of parameterization errors, that can result in a mis-behaving or even completely in-operable transport stack;
- to improve the performance of TP4 when recovery from lost packets becomes necessary;

Since a widely deployed, well tested algorithm is known for an adaptive retransmission timer, we recommend to also deploy it within the ATN for the reasons given above.

However, to derive a robust algorithm, the binary back-off approach (i.e. Karn's algorithm) described above must also be added to the algorithm.

4 Change Proposal for SARPs text

The following text should be added to section 5.5.2.2:

The value of the local retransmission timer T1 shall be greater than the round-trip time (i.e. the time between transmission of a DT-TPDU, and the reception of the associated ACK-TPDU) of the connection that is supervised.

Note: - Choosing a value of T1 that is greater than the round-trip time under all circumstances is a difficult task. Therefore, in practice one might chose a value of T1 that is two to three times the average round-trip time that is expected on the connection being supervised.

Recommendation. - To avoid the need to manually configure the value of T1, an adaptive local re-transmission timer should be implemented. This can be achieved if the sending transport entity continuously measures the round-trip time of DT-TPDUs it transmits. Based upon these sample values, a smoothed estimate of the average round-trip time can be computed according to

$$error := measuredRTT - estimatedRTT$$
$$estimatedRTT := estimatedRTT + \alpha \cdot error$$

The sending transport entity must ignore measures from DT-TPDUs that had to be retransmitted. With each new valid sample value, the local retransmission timer T1 finally can be computed according to

$$estimatedDev := estimatedDev + \gamma \cdot (|error| - estimatedDev)$$
$$T1 := estimatedRTT + \zeta \cdot estimatedDev$$

with α, γ, and ζ chosen according to table 5.5.2-1. The initial value of *estimatedRTT* can be set to the time between the transmission of a CR-TPDU, and the reception of the associated CC-TPDU. The initial value of *estimatedDev* should be set to the initial value of *estimatedRTT*.

Table 5.5.2-1: Adaptive retransmission timer values

Name	Description	Recommended value
α	Time constant of filter to compute a smoothed estimate of the average round-trip time	1/8
γ	Time constant to filter the estimated deviation of the averaged round-trip time	¼
ζ	Factor that determines "distance" of T1 from average round-trip time computed	1/8

Note: The sending transport entity is not required to measure the round-trip time of each DT-TPDU transmitted. However, this improves the quality of the average round-trip time estimate, and thus reduces the probability of false alarms.

5 References

[1] Van Jacobsen
Congestion Avoidance and Control
ACM SIGCOMM'88, vol.18, no.4, August 1988

[2] Phil Karns, Craig Partridge
Improving Round Trip Time Estimates in Reliable Transport Protocols
ACM SIGCOMM'87, vol. 17, no.5, October 1987

[3] David Mills
Internet Delay Experiments
RFC 889

6 List of Figures