Congestion Control in Broadband Wireless

Lecture 4

Subir Varma

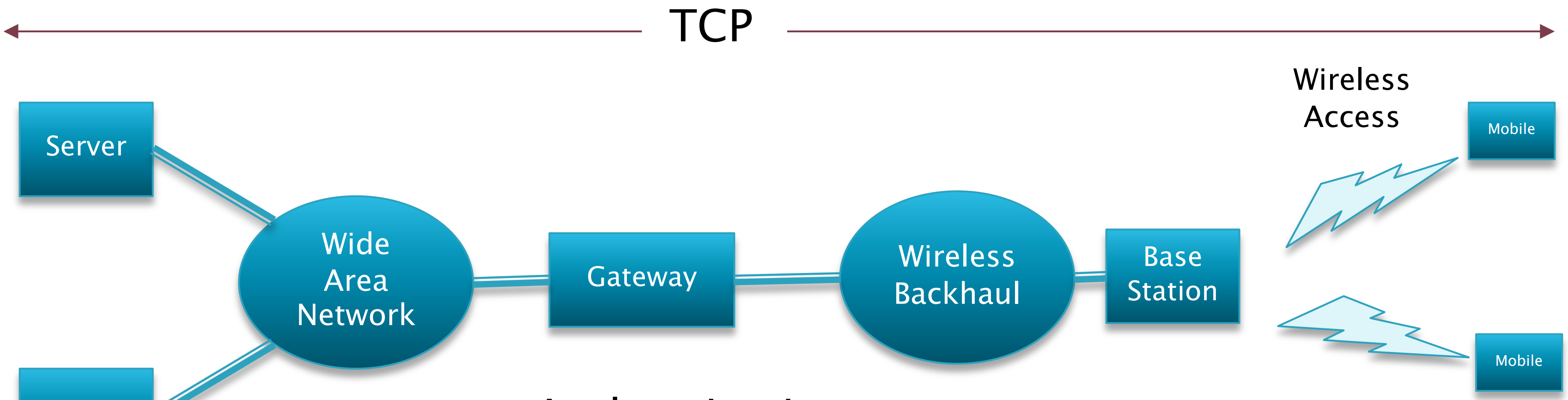
Wireless Networks

- Common examples of broadband wireless networks deployed today include WiFi networks that operate in the unlicensed frequency bands, and broadband cellular networks such as 2G, 3G or LTE that operate in service provider owned licensed bands.

- When TCP is used as the transport protocol over these networks, it runs into a number of problems that are not commonly found in wireline networks.

- Solving problems with wireless transmission of data:
  - Improving the wireless physical layer with more robust modulation and more powerful error correction schemes
  - Advances in the Medium Access Control(MAC) Layer.
  - Modifying the congestion control algorithm in order to overcome wireless related link impairments

Cellular Wireless Architecture



Authentication
Cellular Handoff

Wireless MAC
OFDMA

Issues with TCP over Wireless Networks

- Higher Link Error Rates: This has traditionally been the biggest problem with wireless networks, and is caused due to difficult propagation conditions in the air, leading to high link attenuations and multi-path creating reflections from surrounding objects. With the improvement in modulation and network coding techniques over the last 20 years, the error rates have decreased significantly, but are still much higher compared to wired transmission mediums.

- High Link Latency: Higher Link latency in access networks is partially caused due to the physical layer technology used to overcome wireless medium related transmission problems. Round trip latencies of the order of 50 ms in LTE.

  Causes:
  - Forward Error Correction (FEC)
  - MAC Protocol

Issues with TCP over Wireless Networks

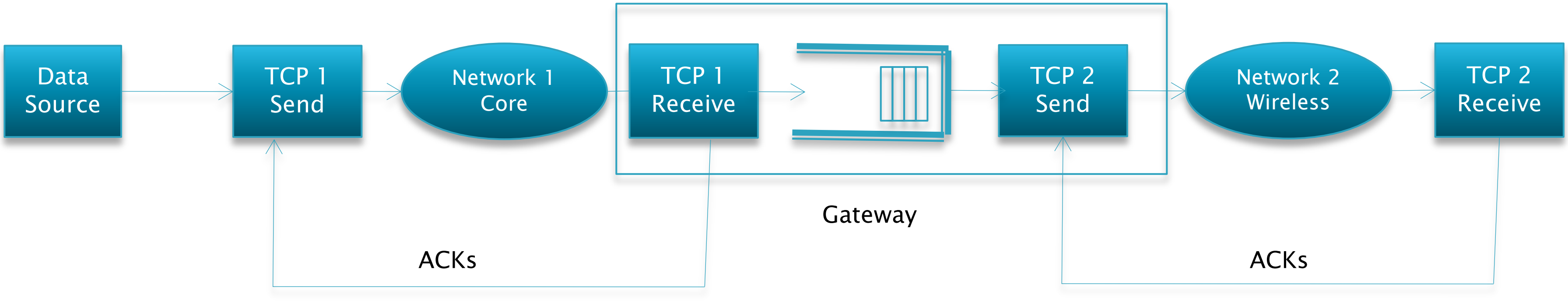
- Large Delay Variation: This is common in wireless cellular networks, and is caused due to the base station adapting its PHY layer transmission parameters such as modulation and coding, as a function of link conditions, on a user by user basis. This causes the effective channel capacity to fluctuate over a wide range, leading to a phenomenon called bufferbloat. This is due to the fact the TCP window can never be perfectly matched with the varying link capacity, and at times when the capacity is low and the window is much bigger, the buffer occupancy increases, as was shown in Lecture 2.

- Random Dis-Connects: This problem can be caused due one of several reasons: For example mobile users are momentarily disconnected from the network when they are doing hand-offs between cells. Alternatively, if the wireless signal fades due to multipath or gets blocked due to an obstruction, then it can cause a temporary disconnect. During this time the TCP source stops receiving ACKs, and if the disconnect time is of the order of 100s of ms, the re-transmission timer expires, thus affecting TCP throughput.

- Asymmetric link capacities: When the uplink has much lower bandwidth compared to the downlink, then the inter-ACK time interval is no longer a good estimate of the bandwidth of the downlink bottleneck capacity hence TCP’s self-clocking mechanism breaks down.

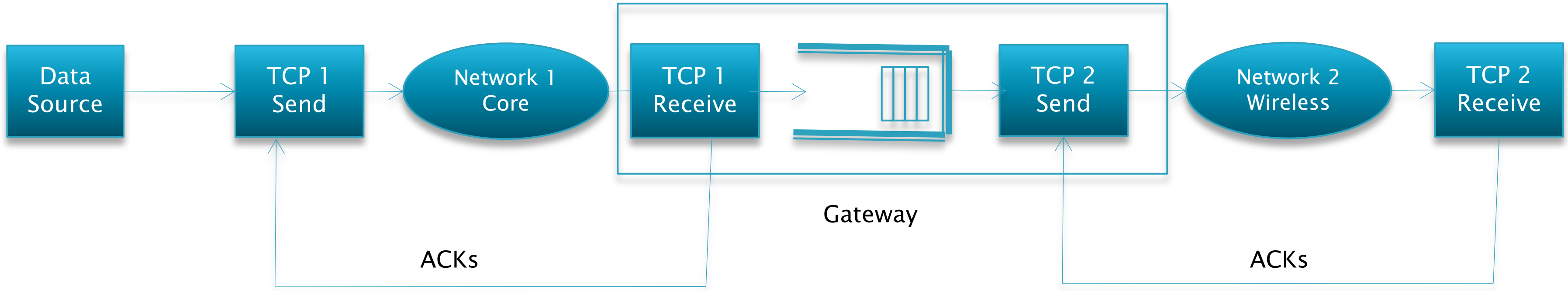
Split Connection TCP

Split Connection TCP



This is not a new congestion control algorithm, but a way to change the structure of the congestion control system so that the problematic part of the network can be isolated. This is done by abandoning the end-to-end nature of the TCP transport layer, in favor of multiple TCP controlled segments in series.

Split Connection TCP



System Operation:

- When the server initiates a TCP connection with a client, the connection set-up message is intercepted by the Gateway, which then proceeds to do the following: (1) Reply to the connection set-up back to the server, as if it were the client, thus establishing the TCP1 connection, (2) Establish the TCP2 connection with the client, with itself as the end-point.

- During the data transfer phase of the connection, the Gateway intercepts all packets coming from the server over TCP1 and immediately ACKs them back. At the same time it forwards the packet on TCP2 to the client, and discards it after it receives the corresponding ACK from the client.

- If the TCP2 is operating at a slower speed than TCP1, then the data buffers in the Gateway start to fill up, and the Gateway then proceed to backpressure the server by reducing the size of the receive window in TCP1's ACK packets.

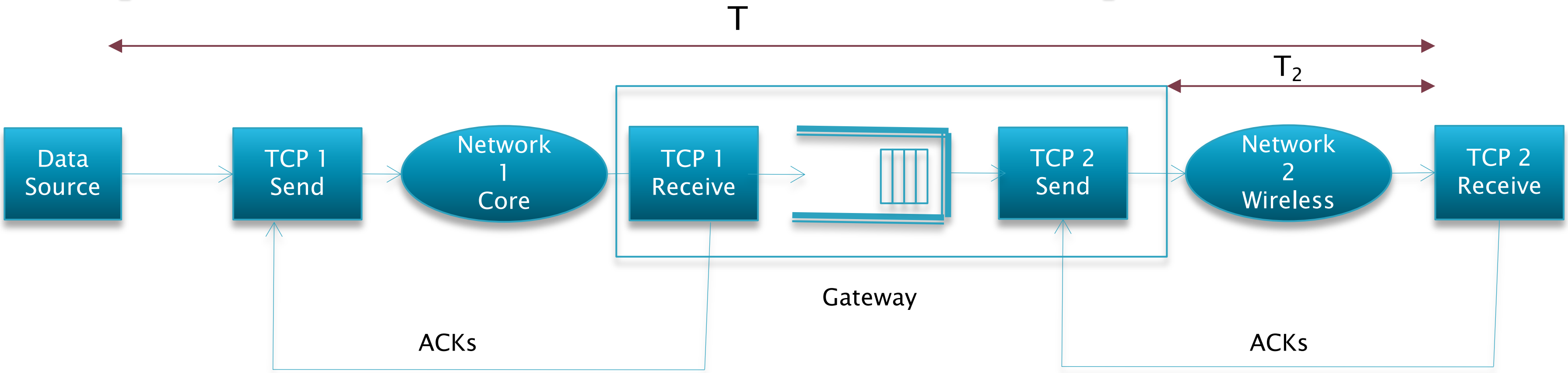
Benefits of Split Connection TCP

- The TCP congestion control algorithms operating on each segment of the network can be tailored to the specific characteristics of the network. The core network is typically constructed from highly reliable optical connections and hence has very low link error rates. The access network on the other hand may present several transmission related problems that were described earlier.

- As a result, the congestion control algorithm over the wireless network can be specially tailored to overcome these link problems. Note that this can be done without making any changes to TCP1, which is typically running on some server in the Internet and hence cannot be modified easily

- Due to the way that TCP throughput varies as a function of the round trip latency and link error rates, the split connection design can result in a significant boost to the throughput, even in the case in which legacy TCP is used on both connections. This claim is proven below.

Split Connection Analysis



R_{ns}: Throughput of the end-to-end non-split TCP connection
R_{sp}: End-to-end throughput for the split-TCP connection
R_1: Throughput of TCP1 in the split-connection case
R_2: Throughput of TCP2 in the split-connection case
T: Round trip latency for the non-split connection
T_2: Round trip latency for TCP2 in the split-connection case
q_2: Error rate for the bottleneck link in the wireless network

$$R_{ns} = \frac{1}{T}\sqrt{\frac{3}{2q_2}}$$

TCP tpt for non-split case

$$R_{sp} = \min(R_1, R_2)$$
$$= R_2$$

TCP tpt for split case

Assuming $R_1 > R_2$

$$R_2 \approx \frac{1}{T_2}\sqrt{\frac{3}{2q_2}}$$

(why approximate?)

So that $$\frac{R_{sp}}{R_{ns}} \approx \frac{T}{T_2} > 1$$

Split Connection Analysis (cont)

- Simply splitting the TCP connection causes the end-to-end throughput to increase (why?)

- TCP1 is not completely immune to problems in the wireless network even under the split-connection design. Hence, it is necessary to introduce additional mechanisms to boost TCP2's performance over and above what TCP Reno can deliver (how?)

- When T and T_2 are approximately equal (as in satellite access networks), then R_{ns} is approximately equal to R_{sp}, that is, there are no performance benefits to splitting the connection. However, equation 5 assumes that plain TCP Reno is being used for TCP2. If instead we modify TCP2 so that it is more suited to the access network, then we may still see a performance benefit by splitting the connection.

Issues with Split Connections

- Additional complexity: Split TCP requires that the gateway maintain per-connection state.
- The end-to-end throughput for split TCP may not show much improvement if the connection over the wireless network is performing badly. Hence split TCP has to go hand in hand with additional mechanisms to boost the performance of TCP2.
- Split TCP violates the end-to-end semantics for TCP because the sending host may receive an ACK for a packet over TCP1 before it has been delivered to the client.

Improving TCP Performance over Lossy Links

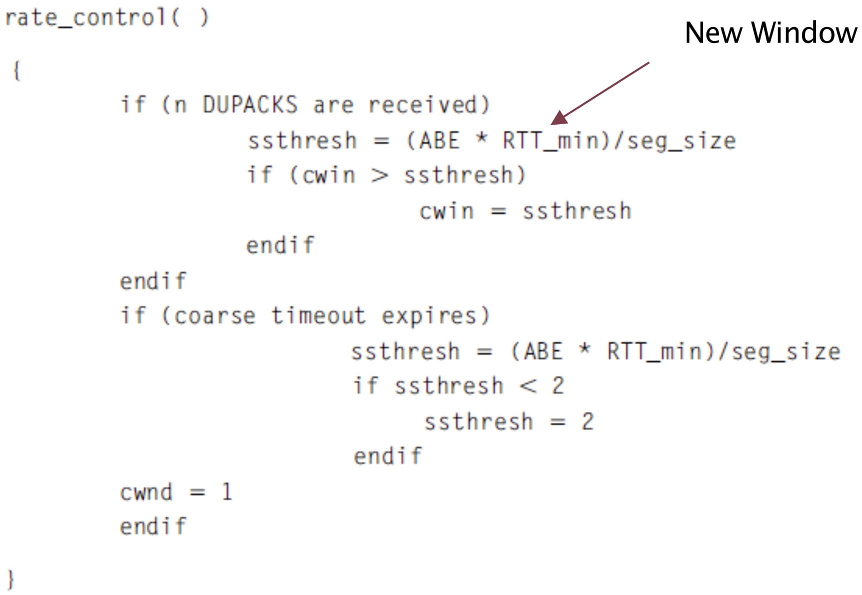
- Modify TCP2 to improve performance over wireless link: Easier to do than modifying end-to-end TCP Reno since TCP2 is under the Operator’s control.

- TCP2 Algorithms:
  - Using available bandwidth estimates (ABEs) to estimate transmit rates.
  - Loss Discrimination Algorithm (LDA)
  - ZRW ACKs

TCP Westwood

TCP Westwood

- The TCP Westwood source attempts to measure the rate at which its traffic is being received, and thus the bottleneck rate, by filtering the stream of returning ACKs.



```
rate_control( )
{
        if (n DUPACKS are received)
                 ssthresh = (ABE * RTT_min)/seg_size
                 if (cwin > ssthresh)
                              cwin = ssthresh
                 endif
        endif
        if (coarse timeout expires)
                        ssthresh = (ABE * RTT_min)/seg_size
                        if ssthresh < 2
                             ssthresh = 2
                        endif
        cwnd = 1
        endif
}
```

New Window Size

- ABE is the estimated connection bandwidth at the source
- RTT_min is the estimated value of the minimum round trip delay
- seg_size is the TCP packet size.

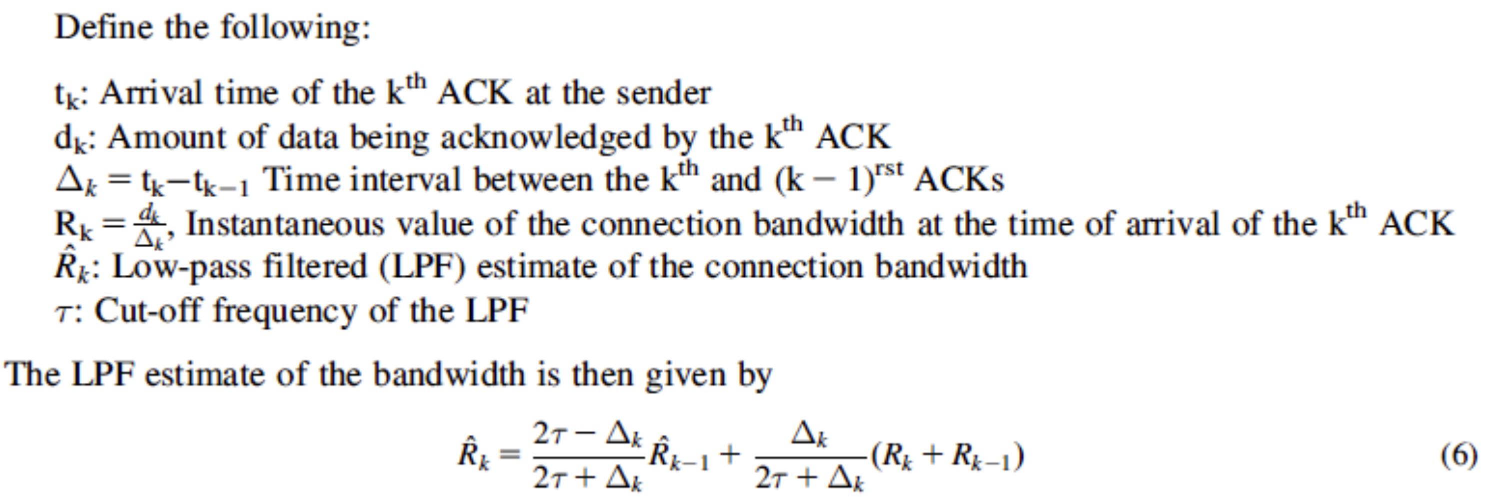
TCP Westwood

Key idea: Set the window size after a congestion event equal to an estimate
of the bottleneck rate for the connection rather than reducing the window size by half as
Reno does.
Hence, if the packet loss is attributable to link errors rather than congestion, then the
window size remains unaffected because the ABE value does not change significantly as a result
of packet drops.

ABE RTT is an estimate of the ideal window siz that the connection should use, given
estimated bandwidth ABE and round trip latency RTT.

However, by using RTT_{min} instead of RTT, Westwood sets the window size to a smaller value,
thus helping to clear out the backlog at the bottleneck link and creating space for other flows.

TCP Westwood: ABE Estimate



Define the following:

t_k: Arrival time of the k^th^ ACK at the sender
d_k: Amount of data being acknowledged by the k^th^ ACK
$\Delta_k = t_k - t_{k-1}$ Time interval between the k^th^ and (k − 1)^rst^ ACKs
$R_k = \frac{d_k}{\Delta_k}$, Instantaneous value of the connection bandwidth at the time of arrival of the k^th^ ACK
$\hat{R}_k$: Low-pass filtered (LPF) estimate of the connection bandwidth
τ: Cut-off frequency of the LPF

The LPF estimate of the bandwidth is then given by

$$\hat{R}_k = \frac{2\tau - \Delta_k}{2\tau + \Delta_k}\hat{R}_{k-1} + \frac{\Delta_k}{2\tau + \Delta_k}(R_k + R_{k-1}) \qquad (6)$$

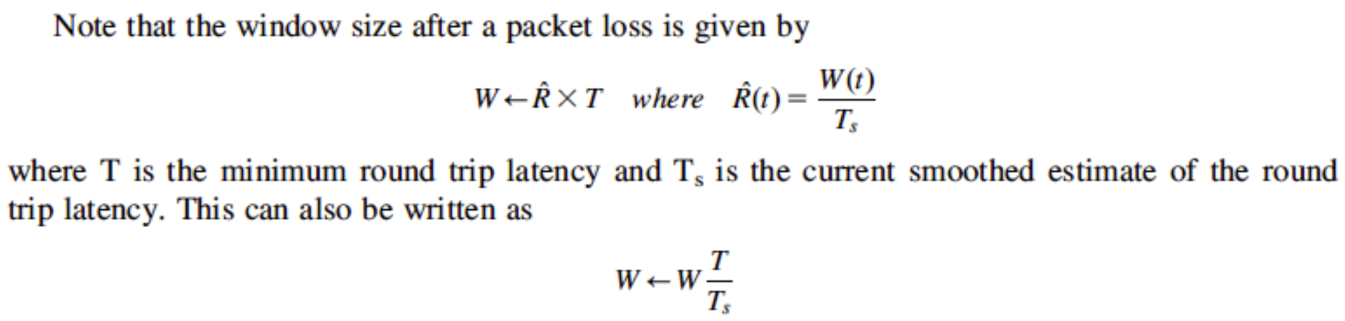
Low-pass filtering is necessary because congestion is caused by the low-frequency components and because of the delayed ACK option.
The filter coefficients are time varying to counteract the fact that the sampling intervals Δk are not constant.

Westwood+: To counteract ACK compression

Instead of computing the bandwidth R_k after every ACK is received, compute it once every RTT seconds. Hence, if D_k bytes are acknowledged during the last RTT interval Δk, then

$$R_k = \frac{D_k}{\Delta_k}$$

Is TCP Westwood AIMD?



Note that the window size after a packet loss is given by

$$W \leftarrow \hat{R} \times T \quad where \quad \hat{R}(t) = \frac{W(t)}{T_s}$$

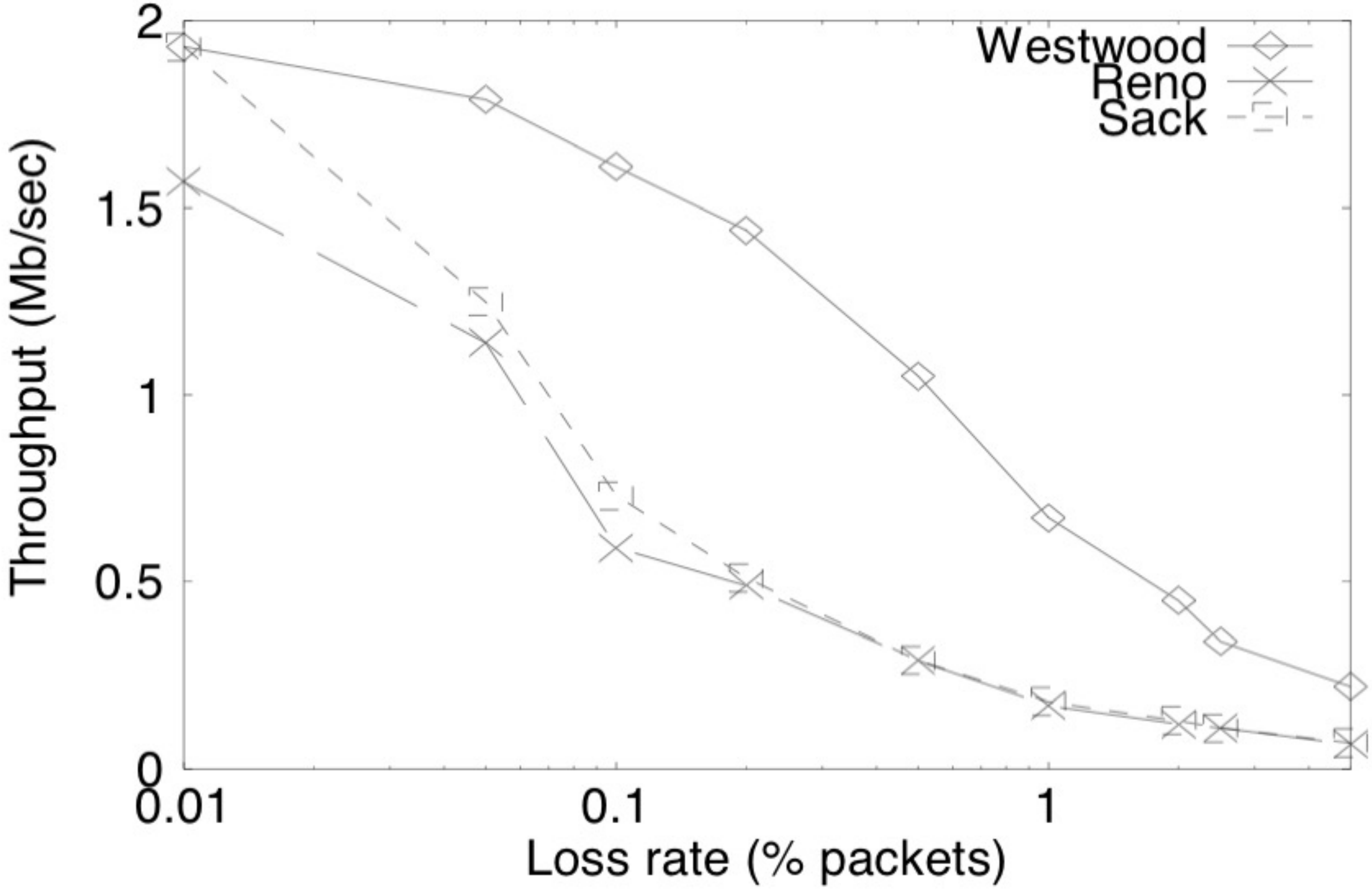
where T is the minimum round trip latency and T_s is the current smoothed estimate of the round trip latency. This can also be written as

$$W \leftarrow W \frac{T}{T_s}$$

- Hence, the change in window size is given by $W(T_s - T)/T_s$.
- This equation shows that Westwood uses a multiplicative decrease policy, where the amount of decrease is proportional to the queuing delay in the network. It follows that TCP Westwood falls in the class of additive increase/multiplicative decrease (AIMD) algorithms
- However, the amount of decrease may be more or less than that of TCP Reno, depending on the congestion state of the network.

What if the actual value of T is less than the estimate at the source?
The algorithm ends up setting the window size W to larger value after a packet loss.

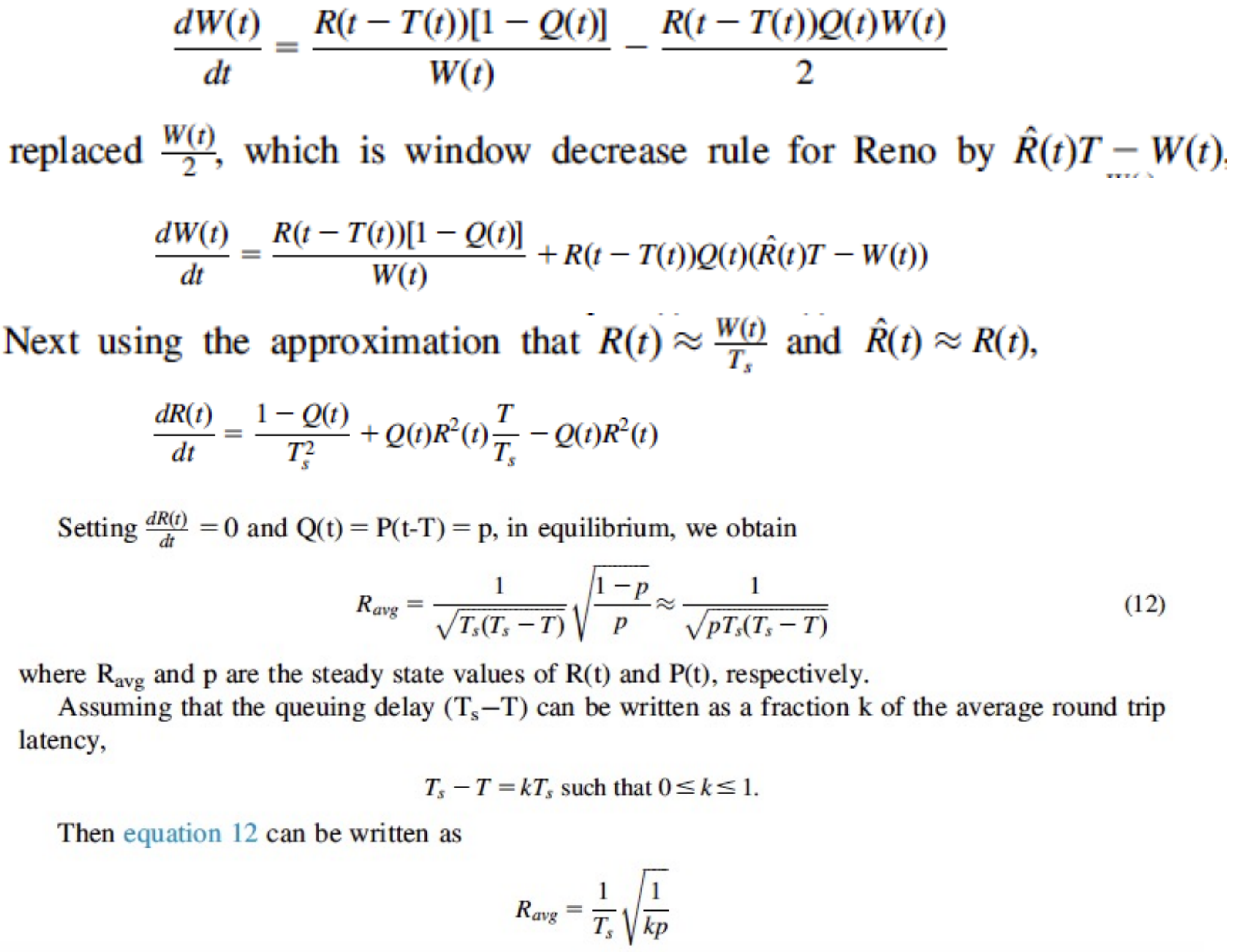
Westwood vs Reno



Westwood gets 16% more bandwidth than its fair share

TCP Westwood: Throughput Analysis

Start with TCP Reno equation for Window Size dynamics:



$$\frac{dW(t)}{dt} = \frac{R(t-T(t))[1-Q(t)]}{W(t)} - \frac{R(t-T(t))Q(t)W(t)}{2}$$

replaced $\frac{W(t)}{2}$, which is window decrease rule for Reno by $\hat{R}(t)T - W(t)$,

$$\frac{dW(t)}{dt} = \frac{R(t-T(t))[1-Q(t)]}{W(t)} + R(t-T(t))Q(t)(\hat{R}(t)T - W(t))$$

Next using the approximation that $R(t) \approx \frac{W(t)}{T_s}$ and $\hat{R}(t) \approx R(t)$,

$$\frac{dR(t)}{dt} = \frac{1-Q(t)}{T_s^2} + Q(t)R^2(t)\frac{T}{T_s} - Q(t)R^2(t)$$

Setting $\frac{dR(t)}{dt} = 0$ and Q(t) = P(t-T) = p, in equilibrium, we obtain

$$R_{avg} = \frac{1}{\sqrt{T_s(T_s - T)}}\sqrt{\frac{1-p}{p}} \approx \frac{1}{\sqrt{pT_s(T_s - T)}} \qquad (12)$$

where R_{avg} and p are the steady state values of R(t) and P(t), respectively.

Assuming that the queuing delay (T_s−T) can be written as a fraction k of the average round trip latency,

$$T_s - T = kT_s \text{ such that } 0 \le k \le 1.$$

Then equation 12 can be written as

$$R_{avg} = \frac{1}{T_s}\sqrt{\frac{1}{kp}}$$

TCP Westwood: Throughput Analysis

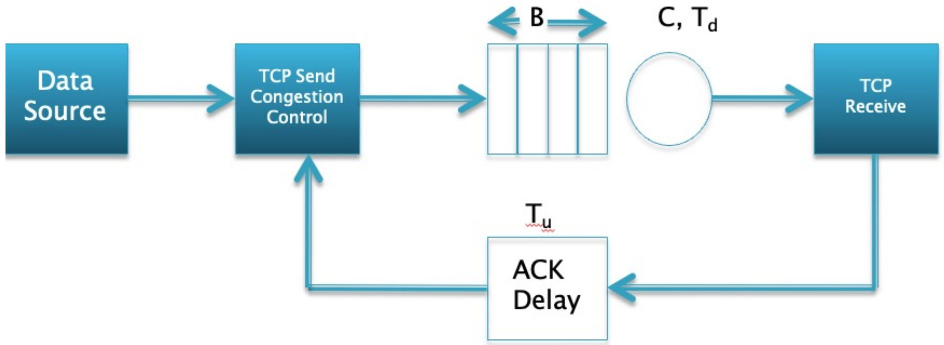
$$R_{avg} = \frac{1}{T_s}\sqrt{\frac{1}{kp}}$$

- On comparing this equation with that for TCP Reno, it follows that TCP Westwood has the effect of a net reduction in link error rate by the fraction k.

- This reduction is greatest when the queuing delay is small compared with the end-to-end latency.

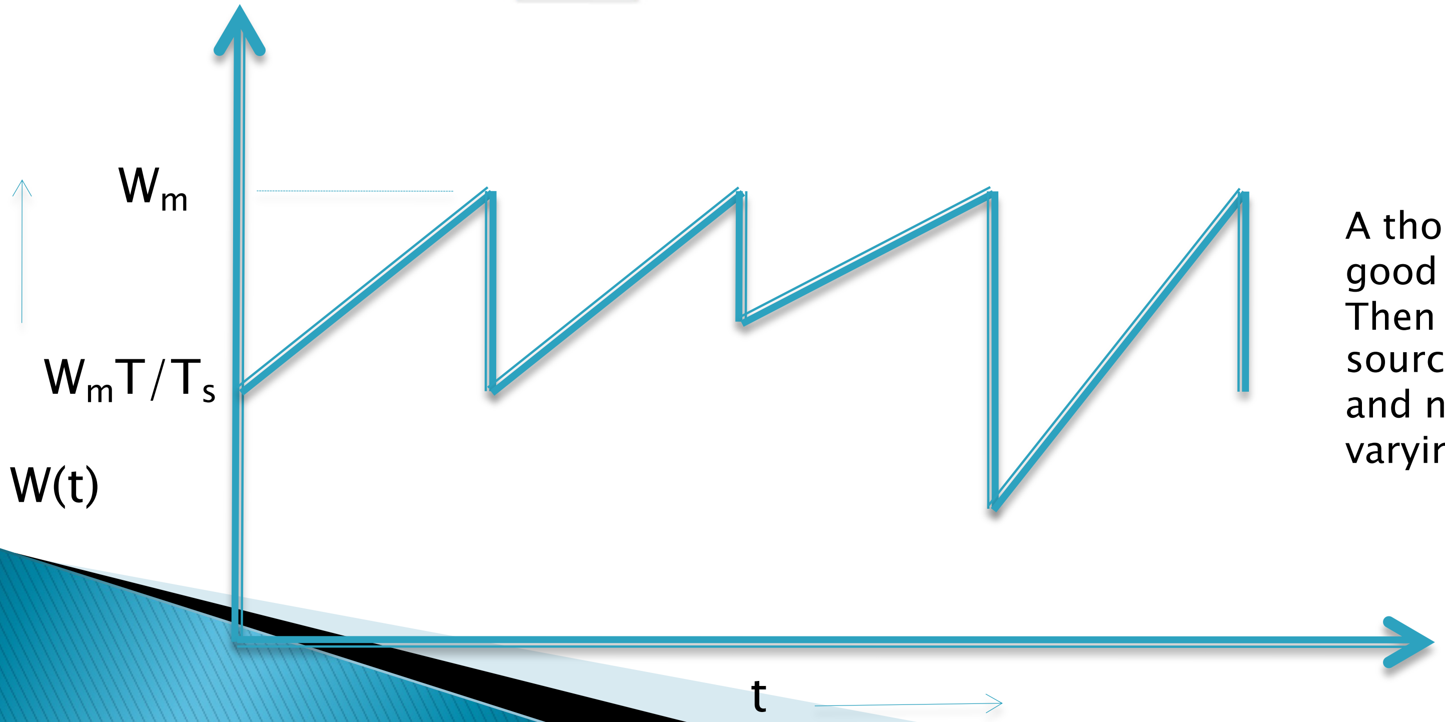
- From this, it follows that Westwood is most effective over long-distance links and least effective in local area network (LAN) environments with a lot of buffering

TCP Westwood



ABE is an estimate of the bottleneck link capacity for the flow.

If there is more than one flow, then ABE varies depending upon the other flows present at the link.



A thought: If we have a good estimate of of ABE, Then why not set the source rate to ABE and not bother with varying the rate at all?

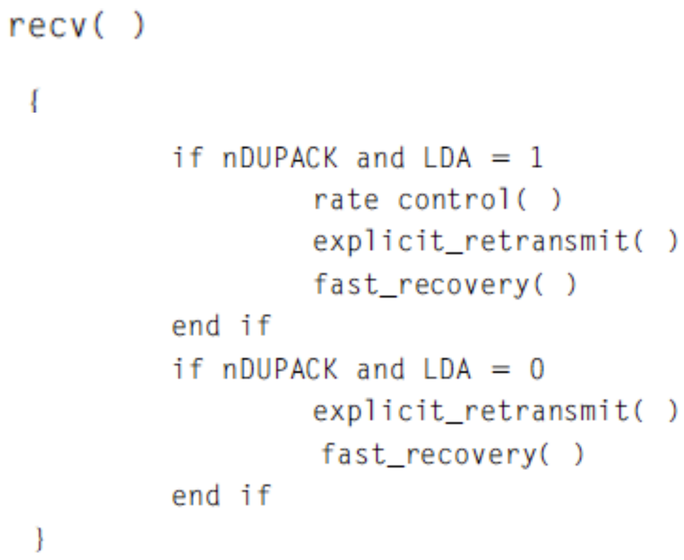
Loss Discrimination Algorithms : LDAs

- Loss Discrimination Algorithms are useful tools that can be used to differentiate between packet losses caused by congestion versus those caused by link errors.
- This information can be used at sender to appropriately react on receiving duplicate ACKs by not reducing the window size if link errors are the cause of the packet drop.
- TCP Veno:
  - Recall from TCP Vegas

$$D(t) = R_E(t) - R(t) = \overset{\text{Max Tpt}}{\frac{W(t)}{T}} - \overset{\text{Estimated Tpt}}{\frac{W(t)}{T_s}} = R(t)[T_s - T] \leftarrow \text{Queue Backlog (Little's Law)}$$

The congestion indicator in the LDA is set to 0 if D(t) is less than 3 and is set to 1 otherwise.
TCP Veno uses this rule in conjunction with Reno's regular multiplicative decrease rule and a slightly modified additive increase rule and observed a big increase in TCP performance in the presence of random errors.

Combining TCP Westwood with LDA



```
recv( )
 {
        if nDUPACK and LDA = 1
                rate control( )
                explicit_retransmit( )
                fast_recovery( )
        end if
        if nDUPACK and LDA = 0
                explicit_retransmit( )
                 fast_recovery( )
        end if
  }
```

Packet loss due to congestion detected

Packet loss due to Link Error detected

Mobile TCP

Zero Receive Window (ZRW) ACKs

- A problem that is unique to wireless is that the link may get disconnected for extended periods, for a number of different reasons. When this happens, the TCP source will time out, perhaps more than once, resulting in a very large total time-out interval. As a result, even after the link comes back up, the source may still be in its time-out interval.

- The can be avoided by putting the TCP source into a state where it freezes all retransmit timers, does not cut down its congestion window size, and enters a persist mode. This is done by sending it an ACK in which the receive window size is set to zero.

- In this mode, the source starts sending periodic packets called Zero Window Probes (ZWPs). It continues to send these packets until the receiver responds to a ZWP with a nonzero window size, which restarts the transmission.

ZRW ACKS: M-TCP Protocol

- M-TCP falls within the category of split-TCP designs, with the connection from the fixed host terminated at the gateway and a second TCP connection between the gateway and the client.
- However, unlike I-TCP, the gateway does not ACK a packet from the fixed host immediately on reception but waits until the packet has been ACK'd by the client, so that the system maintains end-to-end TCP semantics.
- The gateway node is responsible for detecting whether the mobile client is in the disconnect state by monitoring the ACKs flowing back from it on the second TCP connection. When the ACKs stop coming, the gateway node sends a zero window ACK back to the source node, which puts it in the persist mode.
- When the client gets connected again, the gateway sends it a regular ACK with a nonzero receive window in response to a ZWP to get the packets flowing again.
- To avoid the case in which the source transmits a window full of packets, all of which get lost over the second connection, so that there is no ACK stream coming back that can be used for the zero window ACK, Brown and Singh suggested that the gateway modify the ACKs being sent to the fixed host, so that the last byte is left un-ACKed. Hence, when the client disconnect does happen, the gateway can then send an ACK back to the fixed host with zero window indication.

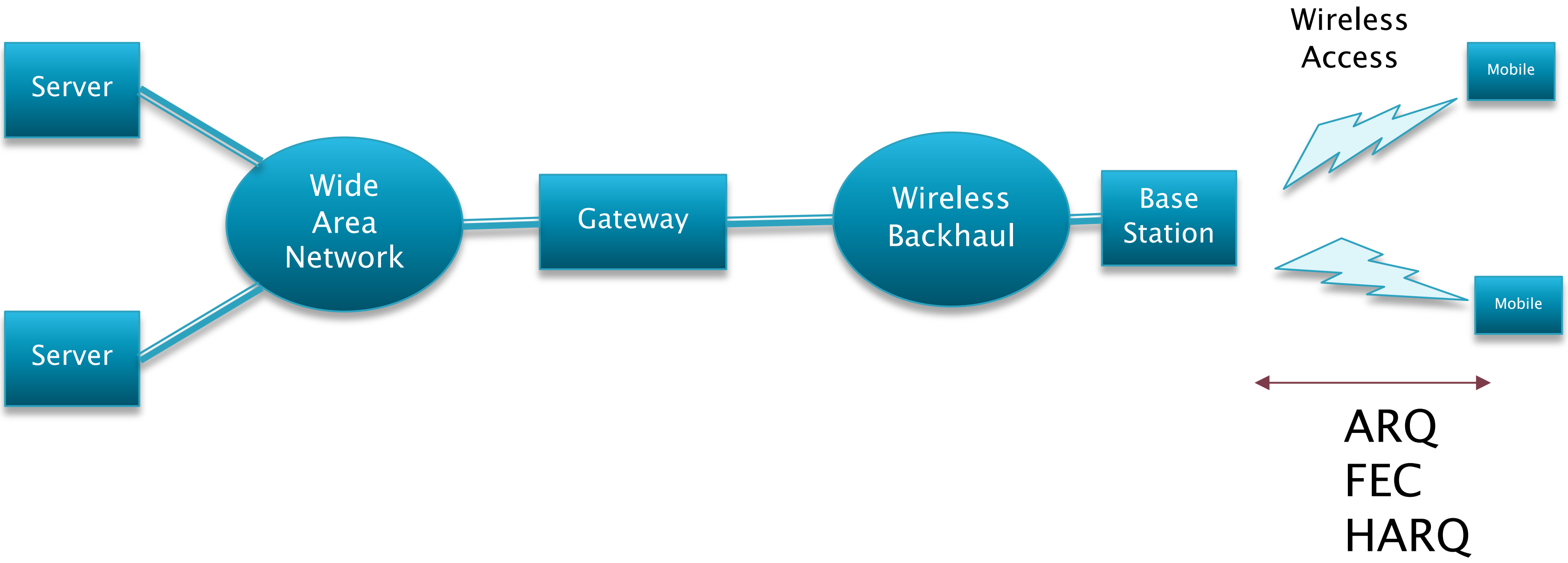
Link Level Error Correction and Recovery

Link Level Error Correction and Recovery

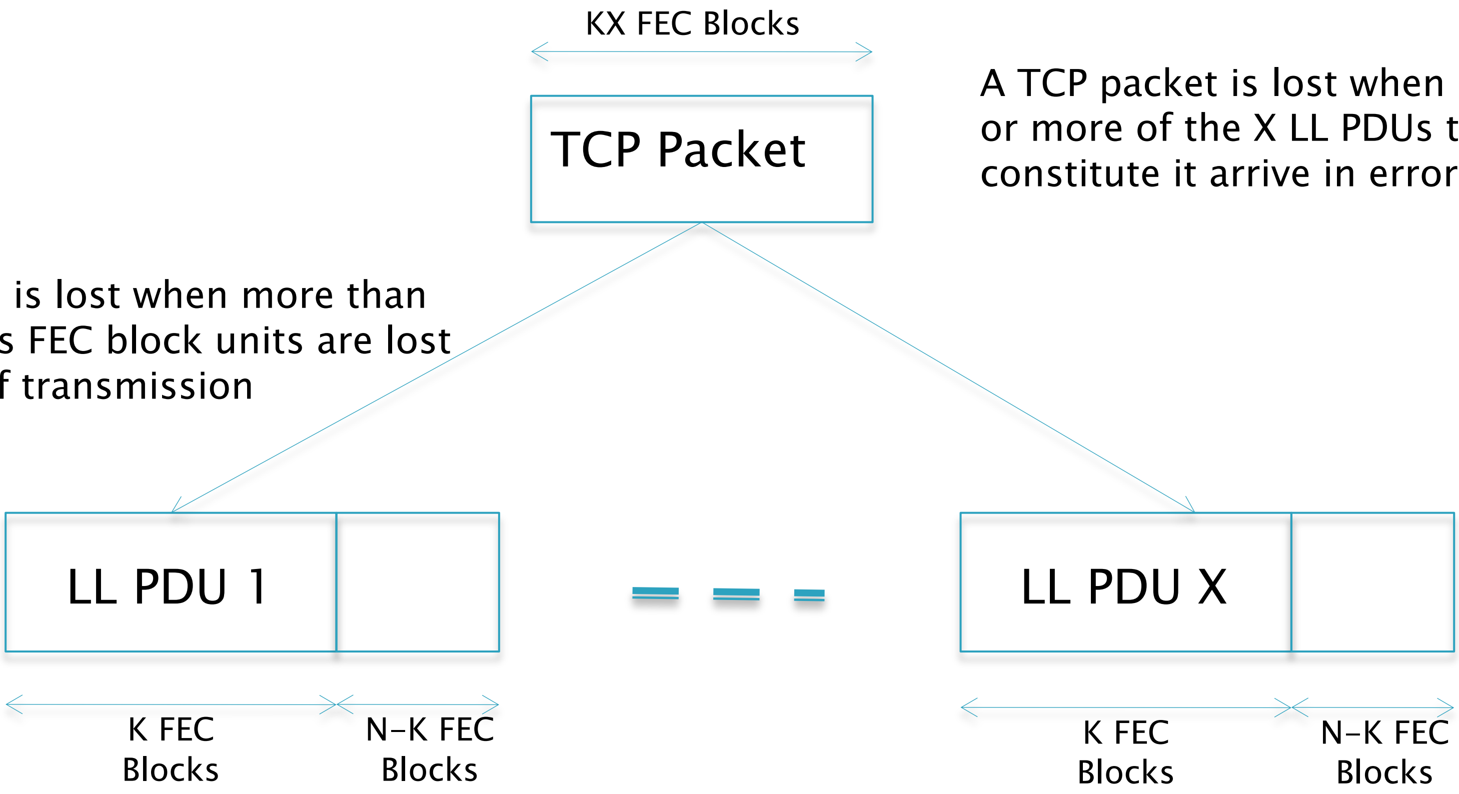
A more direct way of solving the problem of link layer reliability is by reducing the effective error rate that is experienced by TCP. There are several ways this can be done:

- The problem can be attacked at the physical layer by using more powerful error-correcting codes. For example, LTE uses a sophisticated error correction scheme called Turbo Coding to accomplish this. Other error correction schemes in use include Reed-Solomon and Convolutional coding
- Link layer retransmissions or ARQ: As the name implies, the system tries to hide packet loss at the link layer by retransmitting them locally. This is a very popular (and effective) method, widely used in modern protocols such as LTE.
- Hybrid schemes: Both physical layer error correction and link layer retransmissions can be combined together in an algorithm known as Hybrid ARQ (HARQ). In these systems, when a packet arrives in error at the receiver, instead of discarding it, the receiver combines its signal with that of the subsequent retransmission, thereby obtaining a stronger signal at the physical layer. Modern wireless systems such as LTE and WiMAX have deployed HARQ.

Link Level Error Correction and Recovery



Analysis of TCP with FEC

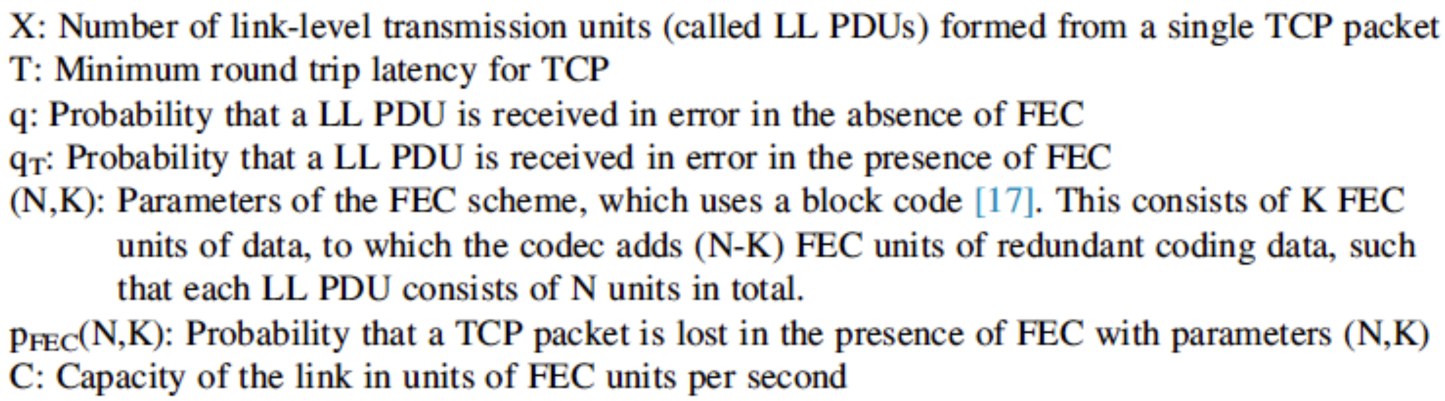


A TCP packet is lost when one or more of the X LL PDUs that constitute it arrive in error

An LL PDU is lost when more than (N–K) of its FEC block units are lost because of transmission errors

Relation between TCP Packets and Link Level PDUs with FEC

Analysis of TCP with FEC



X: Number of link-level transmission units (called LL PDUs) formed from a single TCP packet
T: Minimum round trip latency for TCP
q: Probability that a LL PDU is received in error in the absence of FEC
q_T: Probability that a LL PDU is received in error in the presence of FEC
(N,K): Parameters of the FEC scheme, which uses a block code [17]. This consists of K FEC units of data, to which the codec adds (N-K) FEC units of redundant coding data, such that each LL PDU consists of N units in total.
p_{FEC}(N,K): Probability that a TCP packet is lost in the presence of FEC with parameters (N,K)
C: Capacity of the link in units of FEC units per second

The average TCP throughput in the presence of FEC with parameters (N,K) is given by

$$R_{avg}(N,K) = \min\left(\frac{KX}{T}\sqrt{\frac{3}{2p_{FEC}(N,K)}}, \frac{K}{N}C\right)$$

FEC Blocks/sec

Link Capacity reduces by a factor of K/N

Analysis of TCP with FEC

A TCP packet is lost when one or more of the X LL PDUs that constitute it arrive in error. Hence, it follows that

$$p_{FEC}(N,K) = 1 - (1 - q_T)^X$$

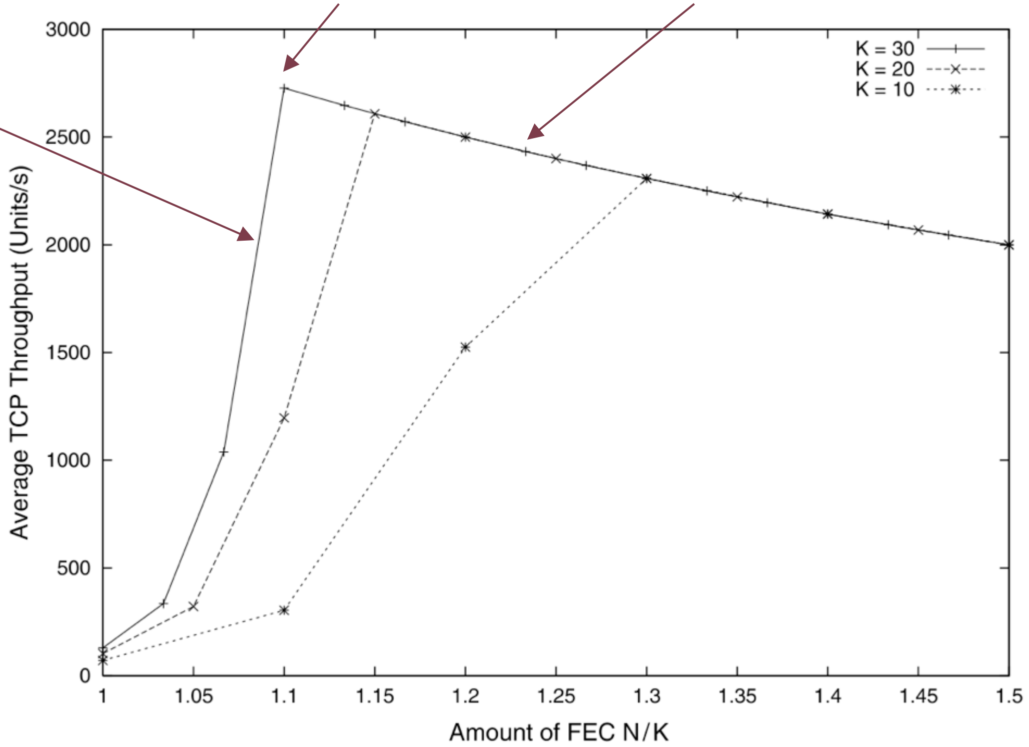
An LL PDU is lost when more than (N-K) of its FEC block units are lost because of transmission errors, which happens with probability

$$q_T = \sum_{i=N-K+1}^{N} \binom{N}{i} q^i (1-q)^{N-i}$$

It follows that the probability that a TCP packet is lost is given by

$$p_{FEC}(N,K) = 1 - \left[1 - \left(\sum_{i=N-K+1}^{N} \binom{N}{i} q^i (1-q)^{N-i}\right)\right]^X$$

TCP Throughput vs N/K for Different Values of K

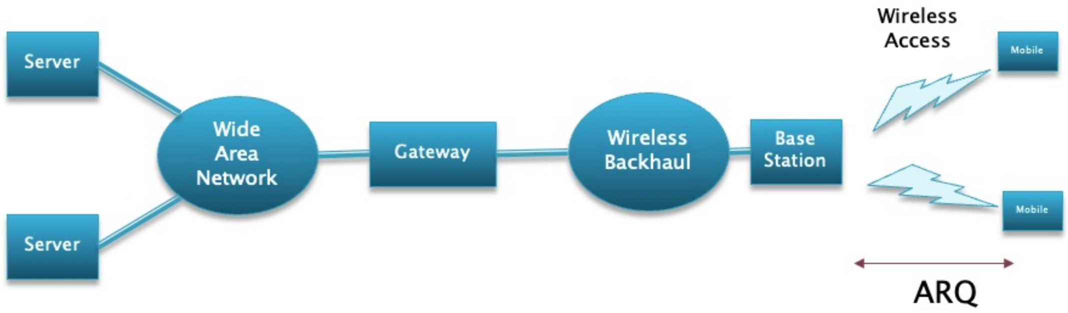


Addition of FEC increases tpt

Optimal amount of FEC

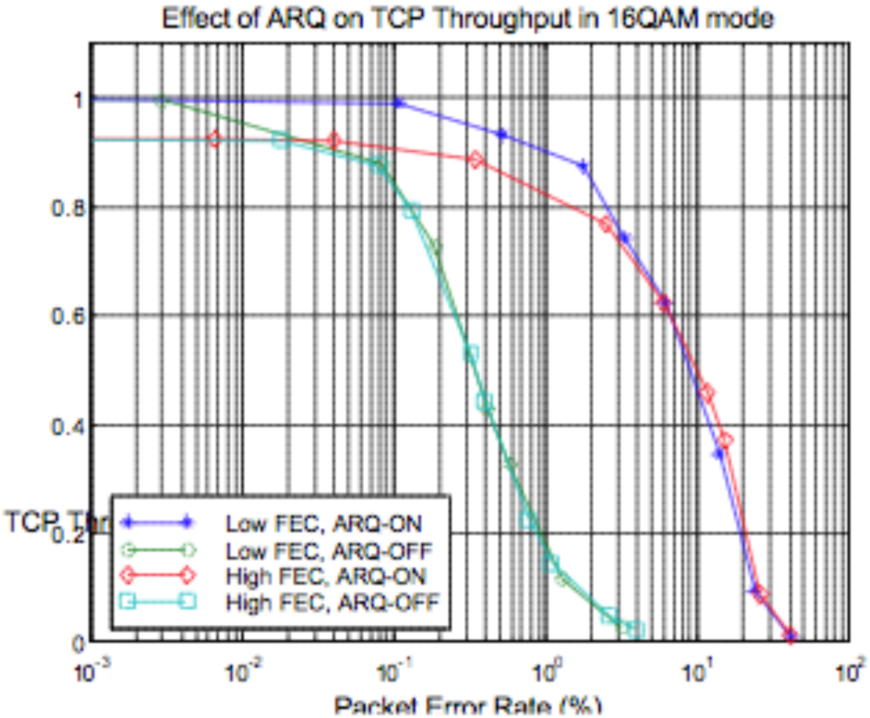
Excess FEC

TCP Throughput in the Presence of FEC and ARQ



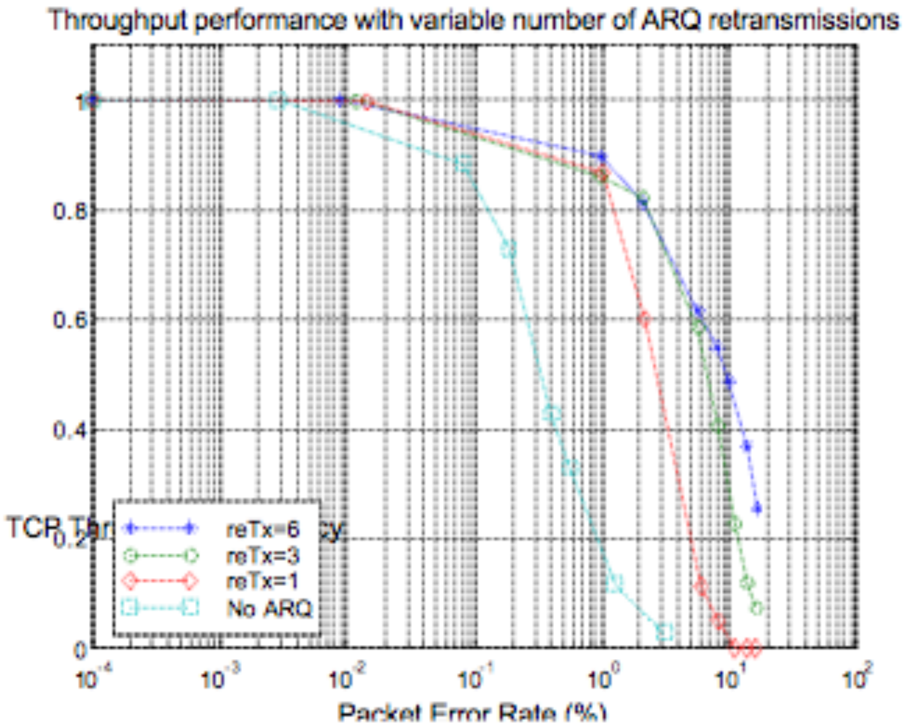
Link-level ARQ or retransmission is an extremely effective way of reducing the link error rate.

TCP Throughput in the Presence of FEC and ARQ



Graph show that even with an error rate as high as 10%, the TCP throughput remains at 40% of the maximum.

Variation of TCP Throughput with Number of ARQ Re-Transmissions



Graph shows that the throughput performance improves as the number of ARQ retransmissions is increased, but after six or so retransmissions, there is no further improvement.
This is because at high error rates, the number of retransmissions required to recover a packet grows so large that the resulting increase in end-to-end latency causes the TCP retransmit timer to expire, thus resulting in a time-out.

Managing Interaction Between TCP and ARQ

Because of its interaction with TCP’s retransmissions, ARQ can in fact make the overall performance worse if not used carefully. Hence, some guidelines for its use include:

- The maximum number of ARQ retransmissions should be capable of being configured to a finite number, and if the LL-PDU is still received in error, then it should be discarded (and let TCP recover the packet). The max number of retransmissions should be set such the total time to do so is significantly less than the TCP retransmit time-out (RTO) interval.
- ARQ should be configurable on a per connection basis, so that connections that do not need the extra reliability can either turn it off or reduce the number of retransmissions.

Controlling Latency

Bufferbloat Problem in Cellular Networks

- Until this point, we have focused on the random packet errors as the main cause of performance degradation in wireless links.
- With advances in channel coding and smart antenna technology, as well as powerful ARQ techniques such as HARQ, this problem has been largely solved in recently designed protocols such as LTE.
- However, packet losses have been replaced by another link related problem, which is the large variability in wireless link latency.

Bufferbloat Problem in Cellular Networks

When there are enough buffers to accommodate the maximum window size W_{max}, the steady-state maximum buffer occupancy at the bottleneck link is given by:

$$b_{\max} = W_{\max} - CT$$

If C varies with time, then one of the following will happen:

- IfC $> W_{max}/T$, then $b_{max} = 0$, i.e., there is no queue build at the bottleneck, and in fact the link is underused
- IfC $<< W_{max}/T$, then a large steady-state queue backlog will develop at the bottleneck, which will result in an increase in the round trip latency if the buffer size B is large. This phenomenon is known as bufferbloat.

Note that buffer size B at the wireless base station is usually kept large to account for the time needed to do ARQ and to keep a reservoir of bits ready in case the capacity of the link suddenly increases. Because C is constantly varying, the buffer will switch from the empty to the full state frequently, and when it is in the full state, it will cause a degradation in the performance of all the interactive applications that are using that link.

Bufferbloat Problem in Cellular Networks

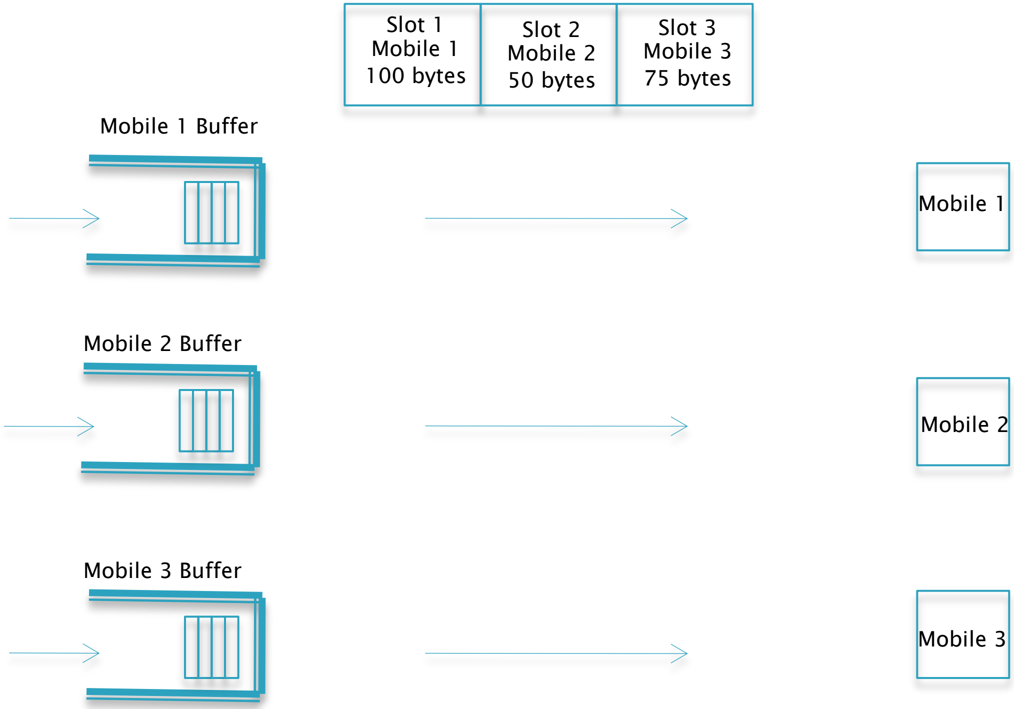
- Note that the rule of thumb that is used in wireline links to size up the buffer size (i.e., B = CT) does not work any longer because the optimal size is constantly varying with the link capacity.
- The bufferbloat problem also cannot be solved by simply reducing the size of the buffer because this will cause problems with the other functions that a base station has to carry out, such as reserving buffer space for retransmissions or deep packet inspections.

Why does C Vary?

There are two main causes for the variation in link capacity:

- ARQ related retransmissions: If the ARQ parameters are not set properly, then the retransmissions can cause the link latency to go up and the capacity to go down.
- Link capacity variations caused by wireless link adaptation:
  - To maintain the robustness of transmissions, the wireless base station constantly monitors the quality of the link to each mobile separately, and if it detects problems, then it tries to make the link more robust by changing the modulation, coding or both.
  - For instance, in LTE, the modulation can vary from a high of 256-QAM to a low of QPSK, and in the latter case, the link capacity is less than 25% of the former.
  - This link adaptation is done on a user-by-user basis so that any point in time different mobiles get different performance depending on their link condition.
  - The base station implements this system, by giving each mobile its own buffer space and a fixed time slot for transmission, and it serves the mobiles on a round robin basis.
  - Depending on the current modulation and coding, different mobiles transmit different amounts of data in their slot, which results in the varying link capacity

Scheduling using Link Adaptation in Cellular Systems



Slot 1
Mobile 1
100 bytes

Slot 2
Mobile 2
50 bytes

Slot 3
Mobile 3
75 bytes

Mobile 1 Buffer

Mobile 2 Buffer

Mobile 3 Buffer

Mobile 1

Mobile 2

Mobile 3

Controlling End-to-End Delay

- How can the end-to-end delay be controlled?
- Approaches that we have seen:
  - End-to-End schemes: These schemes use end-to-end delay measurements to control buffer occupancy at nodes.
    - TCP Vegas
  - Issues: Competition with loss based schemes like Reno, sensitivity to accuracy of measurements.

  - Node based schemes using Active Queue Management (AQM): These schemes try to maintain the buffer occupancy at the node within pre-defined limits.
    - Random Early Detection (RED)
    - Proportional Controller
    - Proportional + Integral Controller
  - Issues: Stable control requires that the Loop Gain be cancelled by a factor that is a function of the link capacity, number of concurrent connections and the round trip latency, all of which are changing with time.

Controlling End-to-End Delay using Explicit Congestion Notification (ECN)

- Once the buffer occupancy exceeds the threshold, the node starts to turn on the ECN bit in passing data packets.
- When these packets reach their destination, the ECN bit is turned on in the corresponding ACKs
- When the source receives packets with the ECN bit turned on, it reduces its transmission rate as if though it detected a dropped packet.
- This allows the source to react to increasing buffer occupancy at the node, without waiting for a packet loss signal due to buffer overflow.

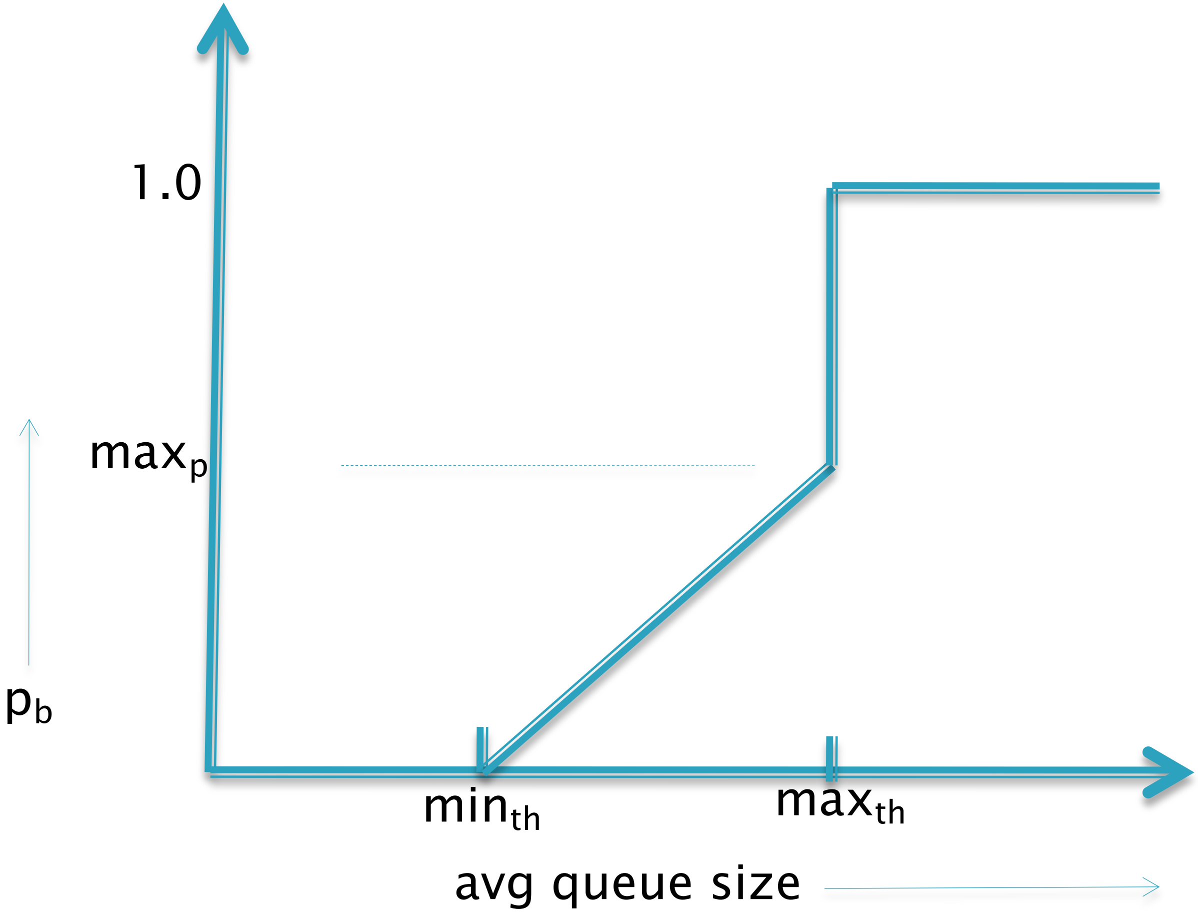
Controlling End-to-End Delay

- This is an area of active research, and in following lectures we will look at several other algorithms that aim to control delay
  - TCP BBR (Bottleneck Bandwidth and RTT) – Lecture 7
  - Improved TCP Vegas type algorithms – Lecture 10
  - Optimization based algorithms – Lecture 11
    These algorithms use an explicit performance objective, which includes end to end delay, to generate their window or rate control rules.

In this lecture we will discuss the following algorithms:

- Active Queue Management (AQM) techniques
  - Adaptive Random Early Detection (A-RED)
  - Controlled Delay (CoDel)
- End-to-End Approaches
  - TCP LEDBAT

The RED Algorithm



$$avg \leftarrow (1 - w_q)avg + w_q q$$

$$p_b \leftarrow \frac{\max_p(avg - \min_{th})}{(\max_{th} - \min_{th})}$$

$$p_a \leftarrow \frac{p_b}{(1 - count \cdot p_b)}$$

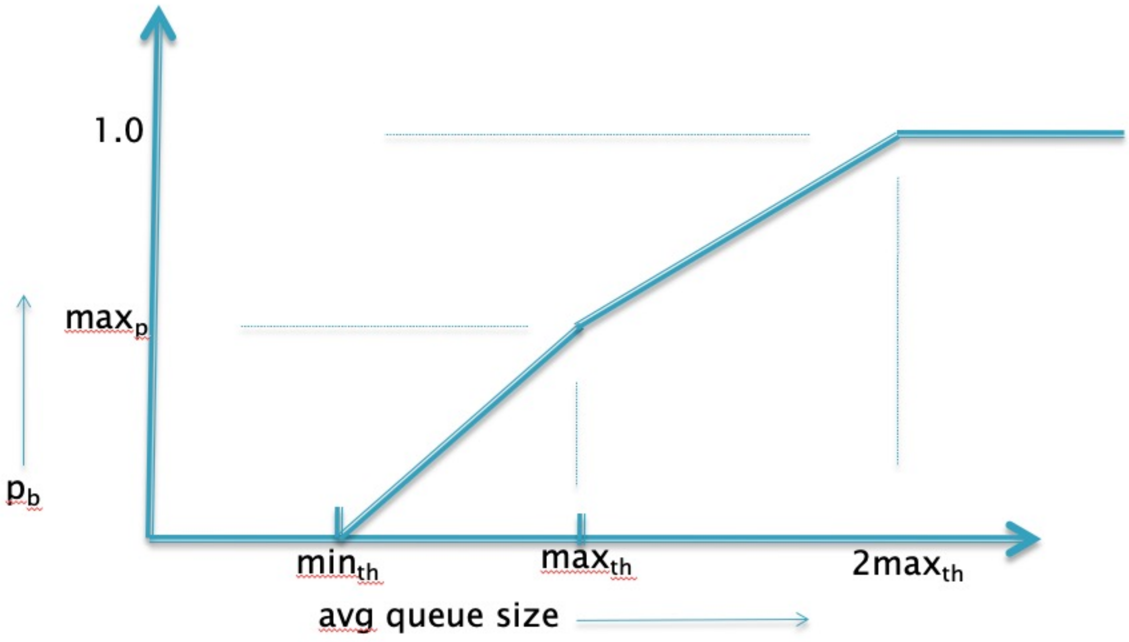
Stability Condition

$$\frac{L_{red}(T^+C)^3}{(2N^-)^2} \leq \sqrt{\frac{\omega_c^2}{K^2} + 1} \quad \text{where}$$

$$\omega_c = 0.1 \min\left\{\frac{2N^-}{(T^+)^2C}, \frac{1}{T^+}\right\}$$

Requires selection of min$_{th}$, max$_{th}$, max$_p$ and w$_q$

Adaptive Gentle RED



Main Idea: Vary max_p to control drop probability

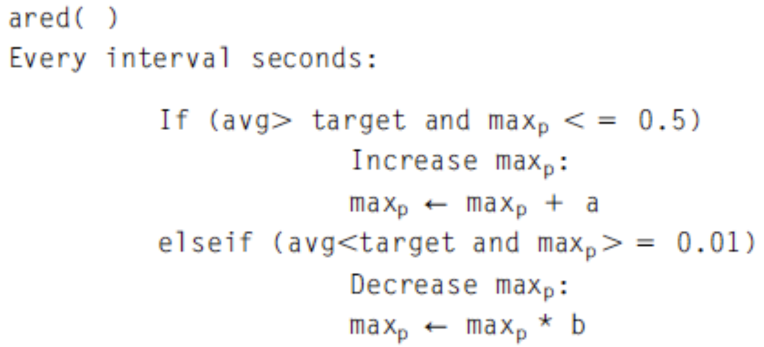
$max_{th} = 3*min_{th}$

$$avg \leftarrow (1 - w_q)avg + w_q q$$

$$w_q = 1 - e^{-\frac{1}{C}}$$

Adaptive Gentle RED

An AIMD algorithm to vary max_p

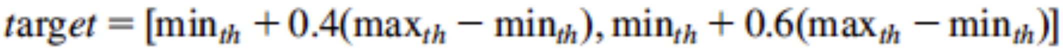


```
ared( )
Every interval seconds:

        If (avg> target and $max_p <= 0.5$)
                Increase $max_p$:
                $max_p \leftarrow max_p + a$
        elseif (avg<target and $max_p >= 0.01$)
                Decrease $max_p$:
                $max_p \leftarrow max_p * b$
```

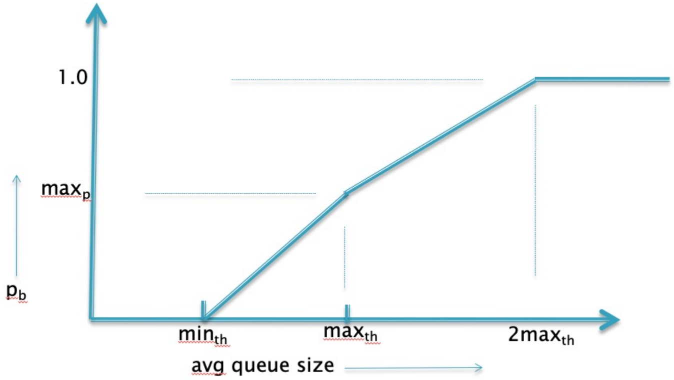
If average queue size exceeds target, then increase max_p

If average queue size is below target, then decrease max_p



$$target = [\min_{th} + 0.4(\max_{th} - \min_{th}), \min_{th} + 0.6(\max_{th} - \min_{th})]$$

interval = 0.5 sec
$a = \min(0.01, max_p/4)$
$b = 0.9$



Controlled Delay (CoDel) Algorithm

- The main innovation in CoDel is to use the packet sojourn time as the congestion indicator rather than the queue size.
- This comes with the following benefits:
  - Unlike the queue length, the sojourn time scales naturally with the link capacity. Hence, whereas a larger queue size is acceptable if the link rate is high, it can become a problem when the link rate decreases. This difference is captured by the sojourn time.
  - Hence, the sojourn time reflects the actual congestion experienced by a packet, independent of the link capacity.

Controlled Delay (CoDel) Algorithm

- CoDel uses the Minimum Sojourn Time as a measure of delay at a node
  - This was chosen in order to distinguish between persistent congestion and transient congestion. CoDel takes action only for persistent congestion.
  - The minimum sojourn time is tracked over an interval equal to the maximum round trip latency over all connections using the link.

- Choose an appropriate value for the Minimum Sojourn Time threshold at Node, such that if the Minimum Sojourn Time is above threshold, then take action to reduce it.

- The Sojourn Time threshold should be kept low in order to reduce latency. How low can it go without hurting throughput?

- The sojourn time threshold is set to 5% of the round trip latency
  Justification:

Recall that $$R_{avg} = \frac{0.75}{\left[1 - \frac{\beta}{(1+\beta)^2}\right]} C \quad where \quad \beta = \frac{B}{CT}$$

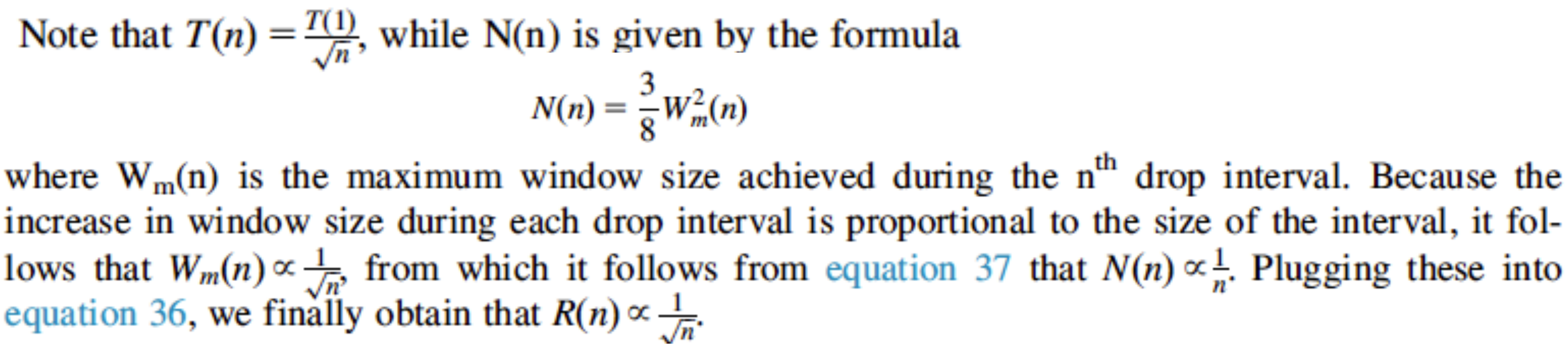
Even at $\beta = 0.05$, $R_{avg} = 0.78C$.
β can be be interpreted as β = (Persistent sojourn time)/T
Hence even at a threshold of 5%, the link utilization is quite high.

Controlled Delay (CoDel) Algorithm

- When the minimum sojourn time exceeds the target value for at least one round trip interval, a packet is dropped from the tail of the queue.

- The next dropping interval is decreased in inverse proportion to the square root to the number of drops since the dropping state was entered. This leads to a gradual linear decrease in the TCP throughput, as can be seen as follows:
  Let N(n) and T(n) be the number of packets transmitted in nth drop interval and the duration of the nth drop interval, respectively. Then the TCP throughput R(n) during the nth drop interval is given by

$$R(n) = \frac{N(n)}{T(n)} \mathrm{n} = 1, 2, \ldots$$



Note that $T(n) = \frac{T(1)}{\sqrt{n}}$, while N(n) is given by the formula

$$N(n) = \frac{3}{8} W_m^2(n)$$

where $W_m(n)$ is the maximum window size achieved during the nth drop interval. Because the increase in window size during each drop interval is proportional to the size of the interval, it follows that $W_m(n) \propto \frac{1}{\sqrt{n}}$, from which it follows from equation 37 that $N(n) \propto \frac{1}{n}$. Plugging these into equation 36, we finally obtain that $R(n) \propto \frac{1}{\sqrt{n}}$.

- The throughput decreases with n until the minimum sojourn time falls below the threshold at which time the controller leaves the dropping state. In addition, no drops are carried out if the queue contains less than one packet worth of bytes.

Design Rules for Wireless Systems

- Compute the quantity $p(CT)^2$ for the system where p is the packet drop rate. If this number is around 8/3, then most of the packet drops are to buffer overflows rather than link errors, so no extra steps are needed to bring down the link error rate.

- If $p(CT)^2 >> 8/3$, then the packet drops are being caused predominantly because of link errors. Furthermore, the link error rate is too high, and additional steps need to be taken to bring it down.
  - If the wireless link supports Reed-Solomon FEC, then use equation 20 to compute p_{FEC}, which is the link error rate in the presence of FEC. If $p_{FEC}(CT)_2 \approx 8/3$, then FEC is sufficient for good performance.
  - If $p_{FEC}(CT)^2 >> 8/3$ then FEC is not sufficient. If the wireless link supports ARQ, then use equation 25 to compute p_{ARQ}, which is the link error rate with both ARQ and FEC. If $p_{ARQ}(CT)^2 \approx 8/3$ for a certain number of retransmissions M, then link-level ARQ+FEC is sufficient for good performance.

Design Rules for Wireless Systems

- If FEC or ARQ are not available on the wireless link or if they do not result in a sufficient reduction in the error rate, then use the Transport layer techniques
  - If the TCP stack at the source is under the designer’s control, then techniques such TCP Westwood with ABE, LDA, or ZRW can be implemented.
  - If the TCP stack at the source is not under the designer’s control, then use a split-level TCP design, with the splitting point implemented at a gateway-type device. In this case, techniques such as Westwood, ABE, or LDA can be implemented on the TCP2 connection.

- If the link error rate is zero but the link shows large capacity variations, then bufferbloat may occur, and the following steps should be taken to limit it:
  - If the base station software can be modified, then AQM schemes such ARED or CoDel should be used.
  - If AQM is not an option, then the delay-based TCP protocols such as Vegas and its variations can be used.

Further Reading

- Chapter 4 of Internet Congestion Control