

The background of the slide is a close-up photograph of a glass surface covered in numerous water droplets of various sizes. Some droplets are large and clear, while others are smaller and more numerous. In the background, out of focus, are several warm-toned lights, likely from a window or interior lighting, creating a bokeh effect with soft, glowing circles. The overall color palette is dark and moody, with the light from the background providing a contrast.

Experimental Investigation of the Google Congestion Control for Real-Time flows

报告人：韩志鹏

目 录

Background

Google Congestion Control

Experimental Testbed

Results

Conclusions

目 录

Background

Google Congestion Control

Experimental Testbed

Results

Conclusions

Background

- Enabling real-time communication over the Internet is a hot topic
- An inter-operable and efficient set of standard protocols is still missing
- IETF RTC Web
 - Aims at standardizing a set of protocols to transport real-time flows.
- W3C Web RTC
 - Aims at standardizing s set of HTML5 APIs to enable real-time communication within Internet browsers.

Background

- Google Congestion Control
 - Proposed by Google within the RTC Web IETF WG.
 - Has already been implemented in the Google Chrome and Firefox browsers.
- Experimental investigation of the GCC
 - Using a controlled testbed which allows bandwidth and propagation times to be set.
 - Investigate to what extent the GCC algorithm is able to
 1. Fully utilize the available bandwidth
 2. Fairly share the bottleneck bandwidth with concurrent flows
 3. Contain queuing delays

目 录

Background

Google Congestion Control

Experimental Testbed

Results

Conclusions

Google Congestion Control

- GCC run over the UDP and it encapsulates the audio/video frames in RTP packets
- Applied only to the video streams since the audio streams bitrate are considered negligible

Google Congestion Control

- Figure 1 shows the main components involved in the architecture
- Two controllers:
 - a sender-side controller
 - Compute the target sending bitrate A_s
 - a receiver-side controller
 - Compute the rate A_r that is sent to the sender under the conditions

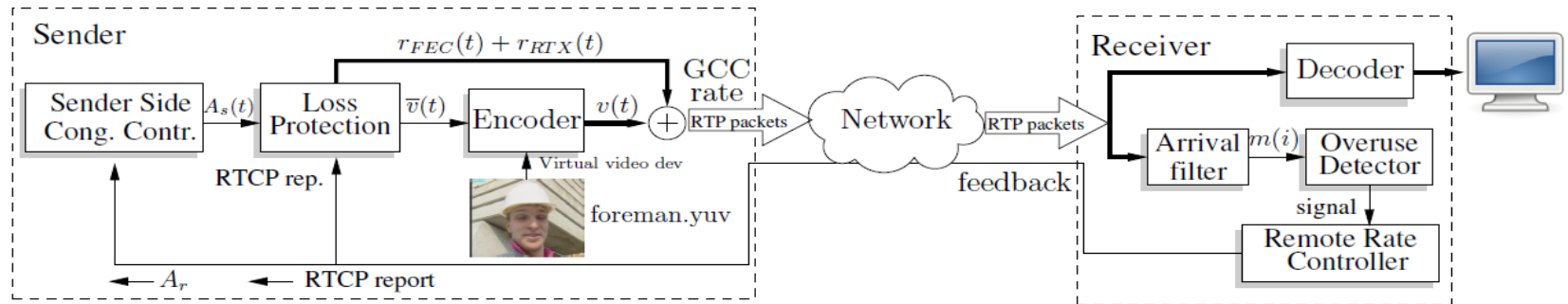


Figure 1: Congestion control architecture

GCC - The sender-side congestion control

- Loss-based congestion control algorithm
 - Every time t_k the k -th RTCP report message arrives at the sender
 - Every time t_r the r -th REMB message, which carries A_r , arrives at the sender
- The frequency RTCP reports are sent is time varying
 - the higher the backward-path available bandwidth, the higher is the RTCP reports frequency.
 - The RTCP reports include the fraction of lost packets $f_l(t_k)$
 - The sender uses $f_l(t_k)$ to compute the sending rate $A_s(t_k)$, measured in kbps

GCC - The sender-side congestion control

- Computation of sending rate $A_s(t_k)$:
 - $X(t_k)$: the TCP throughput equation used by the TFRC(TCP-Friendly Rate Control)

$$A_s(t_k) = \begin{cases} \max\{X(t_k), A_s(t_{k-1})(1 - 0.5f_l(t_k))\} & f_l(t_k) > 0.1 \\ 1.05(A_s(t_{k-1}) + 1\text{kbps}) & f_l(t_k) < 0.02 \\ A_s(t_{k-1}) & \text{otherwise} \end{cases} \quad (1)$$

- When a RMEB is received at time t_r , A_s is set as:

$$A_s(t_r) \leftarrow \min(A_s(t_r), A_r(t_r)).$$

GCC - The receiver-side controller

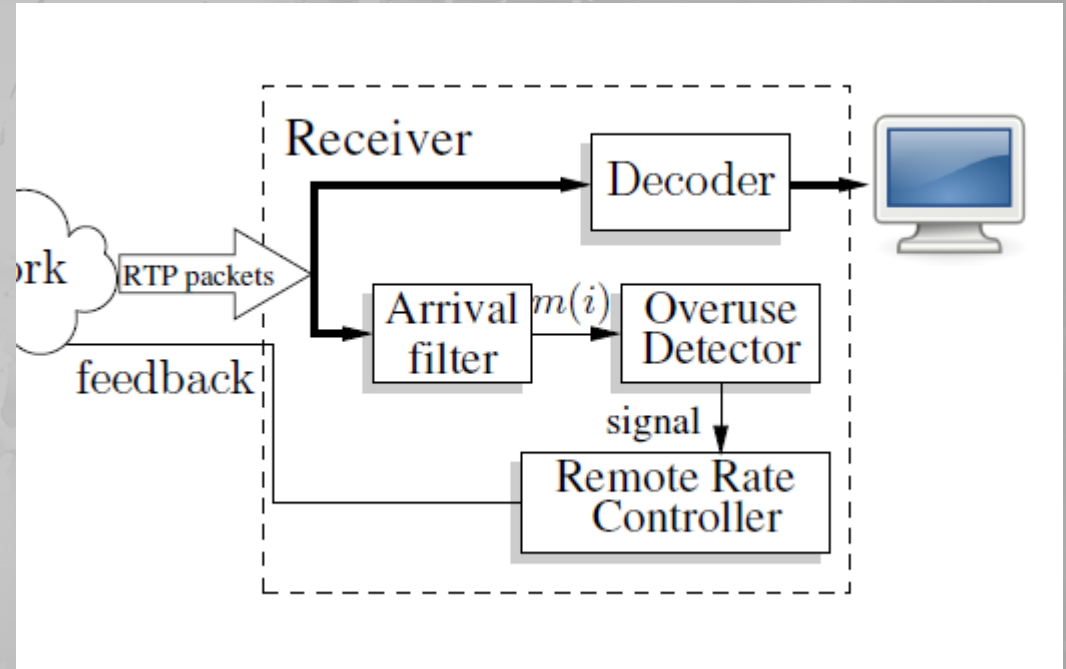
- Delay-based congestion control algorithm
 - Compute A_r according to the following equation:

$$A_r(t_i) = \begin{cases} \eta A_r(t_{i-1}) & \text{Increase} \\ \alpha R(t_i) & \text{Decrease} \\ A(t_{i-1}) & \text{Hold} \end{cases}$$

- t_i : denote the i -th group of RTP packets carrying a video frame is received
- $\eta \in [1.005; 1.3]$, $\alpha \in [0.8, 0.95]$
- $R(t_i)$: the receiving rate measured in the last 500ms

GCC - The receiver-side controller

- Receiver-base controller
 - Arrival-time filter
 - Over-use detector
 - Remote rate controller



GCC - The receiver-side controller

- Receiver-base controller - Arrival-time filter
 - Goal : estimate the queuing time variation $m(t_i)$
 - Measure the one way delay variation:
$$d_m(t_i) = t_i - t_{i-1} - (T_i - T_{i-1})$$
 - T_i is the timestamp at which the i -th video frame has been sent
 - t_i is the timestamp at which the i -th video frame has been received

GCC - The receiver-side controller

- Receiver-base controller - Arrival-time filter
 - One way delay variation is considered as the sum of three components[1]:

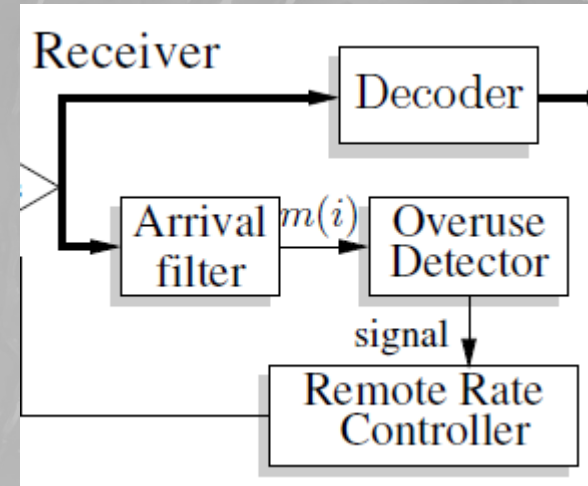
$$d(t_i) = \frac{L(t_i) - L(t_{i-1})}{C(t_i)} + m(t_i) + n(t_i)$$

- $\frac{L(t_i) - L(t_{i-1})}{C(t_i)}$: The transmission time variation.
 - $L(t_i)$: i-th video frame length; $C(t_i)$: estimation of the path capacity
- $m(t_i)$: the queuing time variation
- $n(t_i)$: network jitter which modeled as a Gaussian noise

GCC - The receiver-side controller

- Receiver-base controller - over-use detector
 - Goal : Produce a signal to remote rate controller based on $m(t_i)$
 - Rules :

| Condition | Signal |
|---|-----------|
| if $m(t_i)$ increases above a threshold, and keeps increasing for a certain amount of time, or for a certain amount of consecutive frames | Over-use |
| if $m(t_i)$ decreases below a threshold | Under-use |
| When $m(t_i)$ is close to zero | Normal |



GCC - The receiver-side controller

- Receiver-base controller- remote rate controller
 - Goal : compute A_r by using signal produced by the overuse detector
 - The finite state machine:

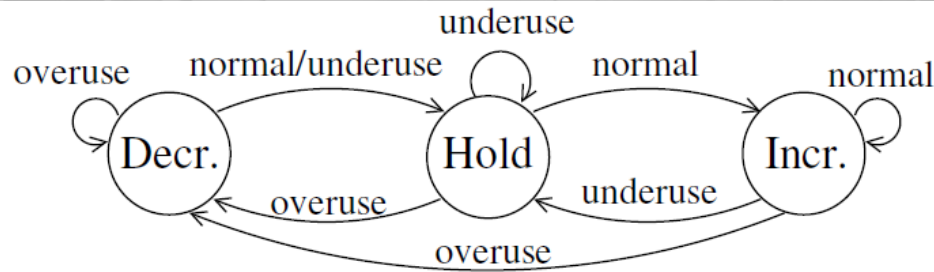
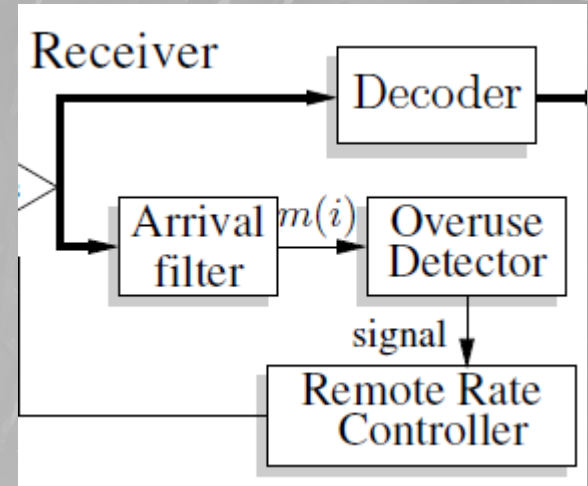


Figure 2: Remote rate controller finite state machine



$$A_r(t_i) = \begin{cases} \eta A_r(t_{i-1}) & \text{Increase} \\ \alpha R(t_i) & \text{Decrease} \\ A(t_{i-1}) & \text{Hold} \end{cases}$$

GCC - The receiver-side controller

➤ Receiver-base controller

- The signaling from the receiver to the sender is done through REMB message carrying A_r or through RTCP reports.
- The frequency at which the REMB messages should be sent in the Google Chrome :
 - If A_r is decreasing, every 1s
 - If A_r decreases more than 3%, immediately

GCC – Loss protection

- Forward Error Correction(FEC)
- Retransmissions to counteract packet losses
- Loss protection is not active

- $v(t) = A_s(t)$

- Loss protection is active

- $\bar{v}(t) = A_s(t) - r_{\text{RTX}}(t) - r_{\text{FEC}}(t)$

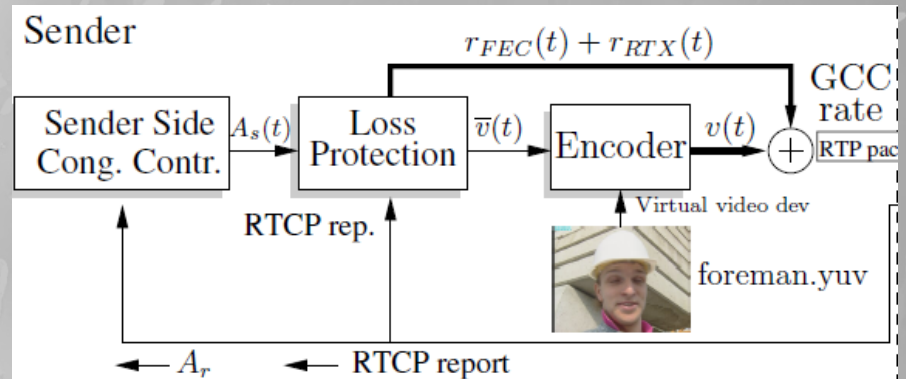
- $r_{\text{FEC}}(t)$:

- sending rate of redundant video frames

- $r_{\text{RTX}}(t)$

- The retransmission rate

- $r_{\text{FEC}}(t)$ can not be more than half of the total sending rate $A_s(t)$



目 录

Background

Google Congestion Control

Experimental Testbed

Results

Conclusions

Experimental Testbed

➤ Testbed

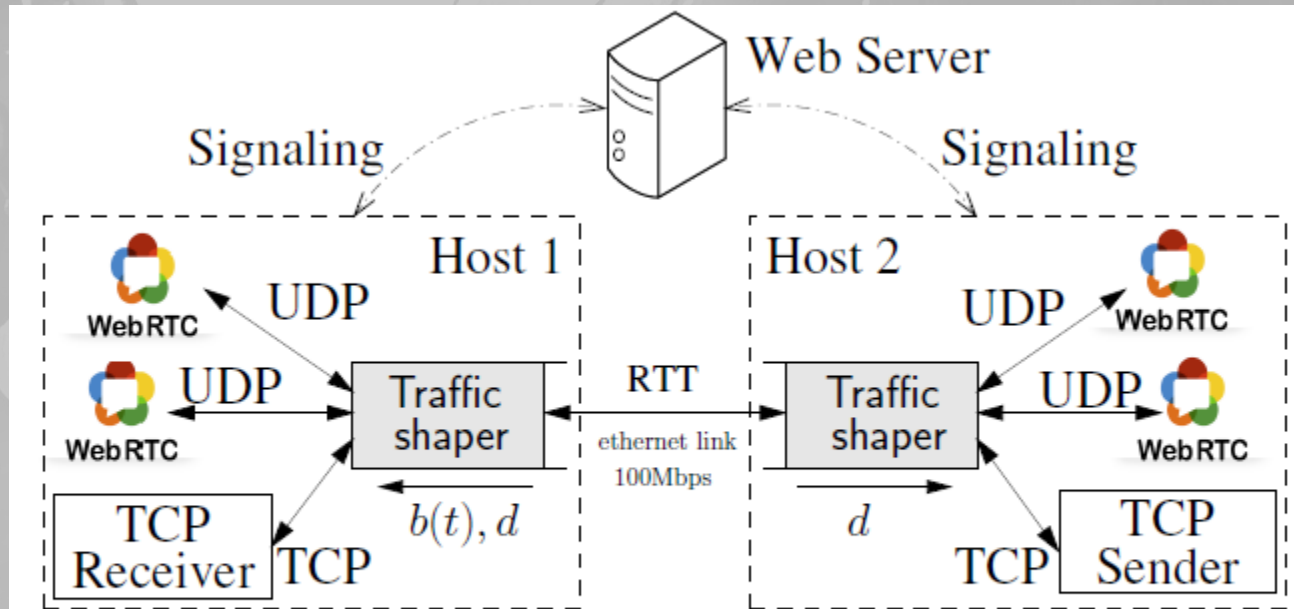
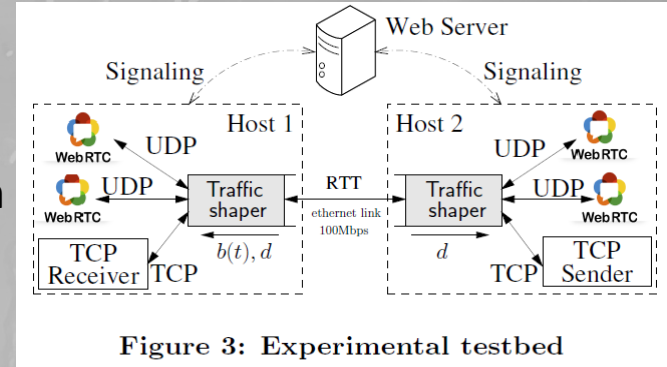


Figure 3: Experimental testbed

Experimental Testbed

- Two host : host1 and host2
- Bottleneck emulating a WAN scenario
 - Traffic shaper tc³ to set delays and bandwidth
- Traffic shaper:
 - Host 1:
 - Set a one-way delay d
 - Set a limitation on the available bandwidth $b(t)$ for the traffic that is received from the Host 2
 - Host 2:
 - Set the one-way delay d for the traffic that comes from the Host 1
- Minimum round trip time : $RTT_m = 2d$



Experimental Testbed

- Buffer size : 60KB
 - Emulate a typical home gateway
- Bandwidth range : [500, 2000]kbps
 - Typical of ADSL uplink speeds
- Both hosts run a chromium browser which generates the video flows
- Host 1 is equipped with a TCP receiver
- Host 2 runs a TCP sender which uses the TCP Cubic congestion control
 - Log the congestion window, slow-start threshold, RTT, sequence number
- Web server provides the HTML pages

Experimental Testbed

- Metrics employed to assess the performance of GCC in the considered scenarios
 - Channel Utilization $U=R/b$:
 - where b is the known available bandwidth and R is the average received rate
 - Good Utilization $g=v/b$:
 - where v is the average video bitrate without considering the bandwidth used for FEC and retransmissions.
 - Loss ratio $l=(\text{lost bytes})/(\text{received bytes})$:
 - it is measured by the traffic shaper tool

目 录

Background

Google Congestion Control

Experimental Testbed

Results

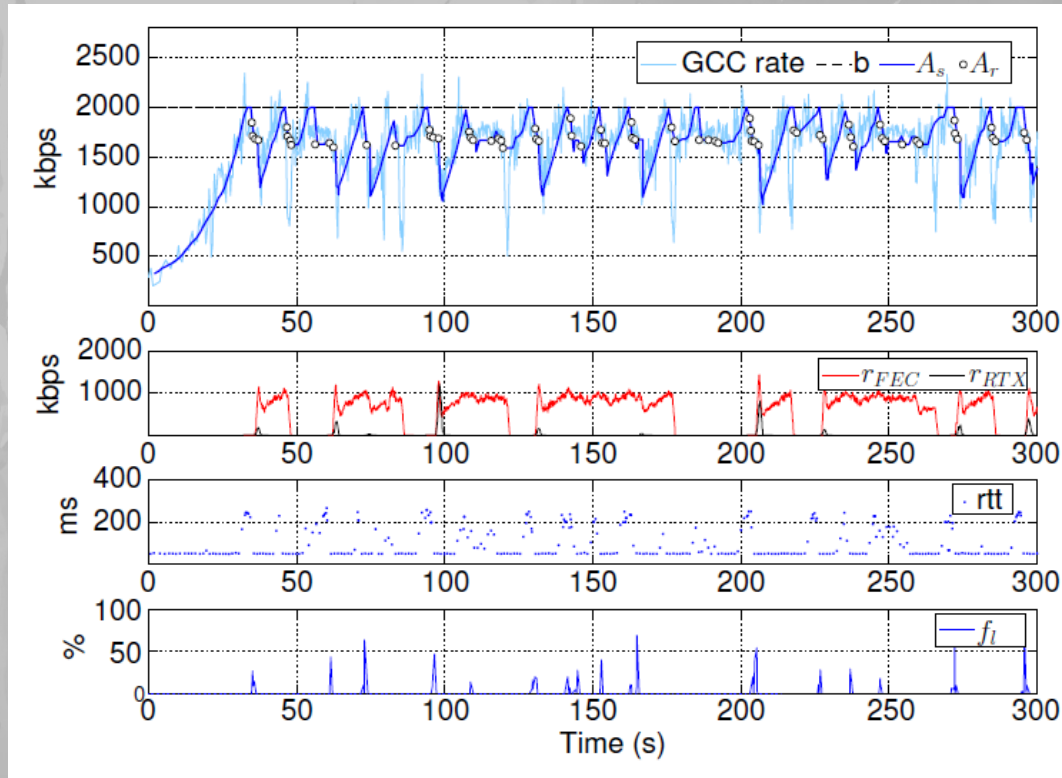
Conclusions

Results

- One GCC flow over a bottleneck with **constant** available bandwidth
- Available bandwidth $b(t) \in \{500, 1000, 1500, 2000\}$ kbps
- Propagation delay $RTT_m = 2d \in \{30, 50, 80, 120\}$ ms
- 16 couples $\{b(t), RTT_m\}$

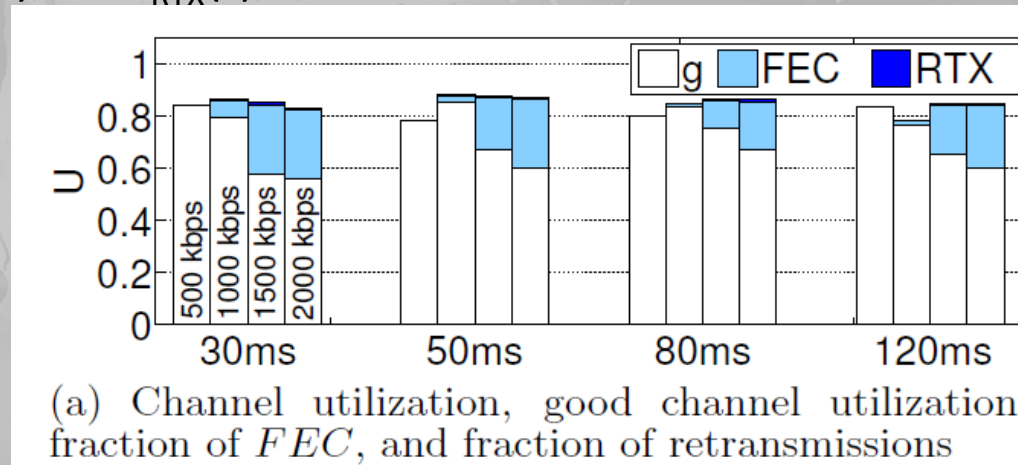
Results

➤ $b=2000\text{kbps}$ and $\text{RTT}_m=50\text{ms}$:



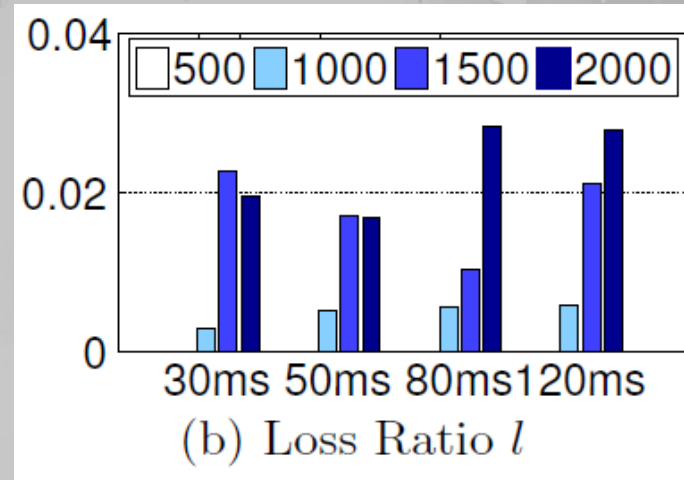
Results

- For each of the considered bottleneck bandwidth b_i and round trip delays RTT_m
- $U = g + r_{FEC}(t)/b + r_{RTX}(t)/b$



- The figure shows that the channel utilization is slightly above 0.8 and is not significantly affected by the RTT_m or the available bandwidth.

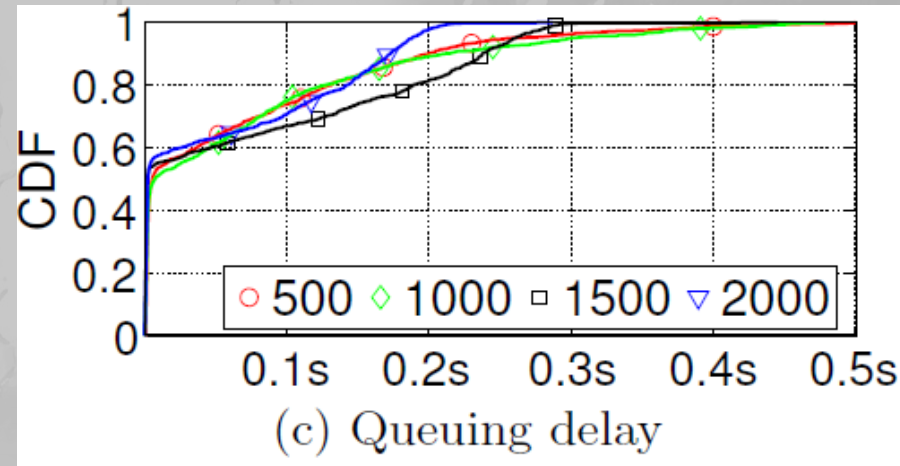
Results



- The figure shows that the loss ratio is not affected by the RTT, but it increases when the bandwidth increases and reaches a maximum value of 0.028.

Results

- We have computed the queuing delay, defined as $Q_{bi}(t) = RTT(t) - RTT_m$
- we have computed the four cumulative distribution function (CDF)



- The figure shows that the median queuing delay is very close to zero, whereas the 90-th percentile is below 0.25s, confirming that the algorithm is able to contain the queuing delay.

Results

- One GCC flow over a bottleneck with **variable** available bandwidth
- Step-like changes of the available bandwidth
- GCC flow accesses a bottleneck whose bandwidth $b(t)$ changes from a minimum value of 400kbps to 3000kbps after 60s.
- Conclusion :
 - GCC reaches a steady-state value of around 2Mbps after a transient time of roughly 30s.

Results

- Let the available bandwidth $b(t)$ vary and the flow adapts the sending rate to it
- Conclusions:
 - b is increased every 100s of 500kbps
- Figure shows that the sending rate is able to quickly match the variable available bandwidth $b(t)$
- the measured channel utilization is slightly above 80%.

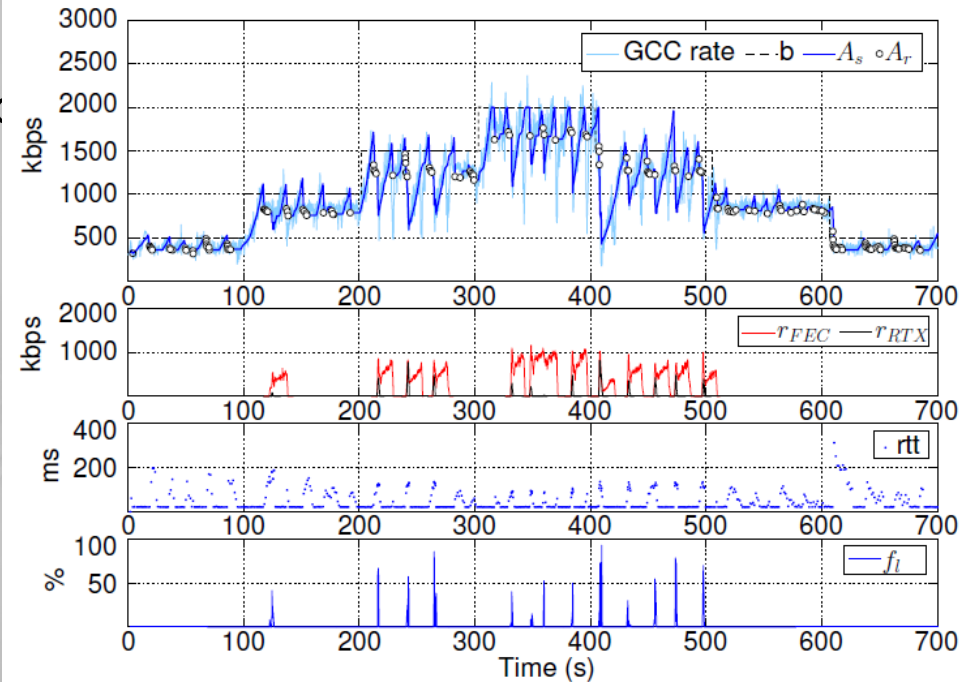


Figure 6: One GCC video flow over a stair-case available bandwidth and $RTT_m = 50\text{ms}$

Results

- One GCC with one concurrent TCP flow
- Three different available bandwidths, $b_i \in \{1000, 2000, 3000\}$ kbps.
- Propagation delay of $RTT_m = 50$ ms.
- Two cases:
 1. A GCC flow starts at $t = 0$ s and a TCP flow enters the bottleneck at $t = 100$ s;
 2. a TCP flow starts at $t = 0$ s and a GCC flow is started at $t = 100$ s.

Results

- Figure shows the overall channel utilization and the TCP and GCC bandwidth shares obtained in this scenario.

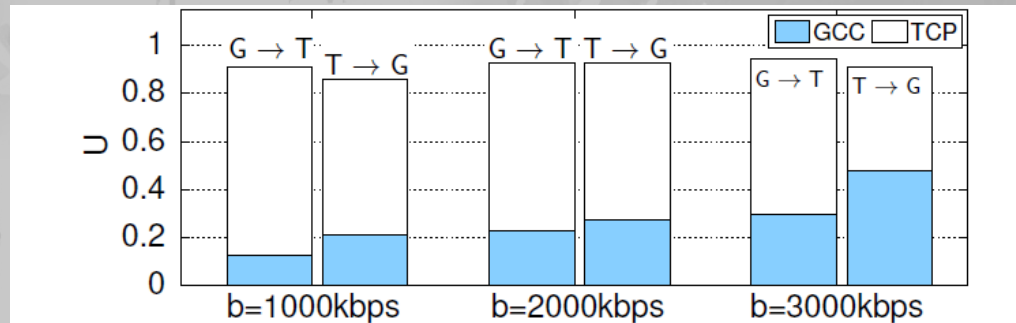


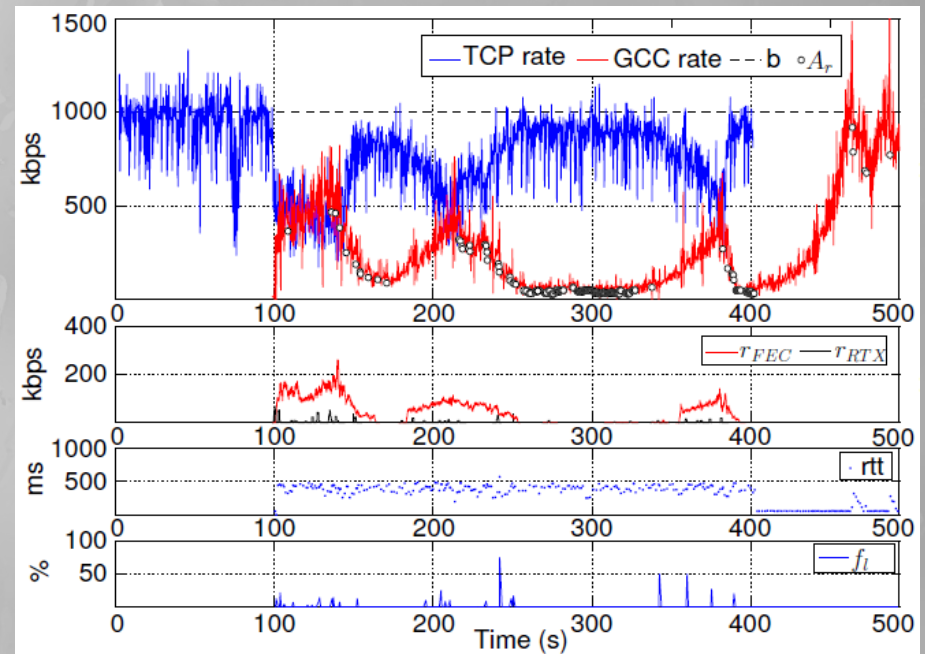
Figure 7: Channel Utilization when one GCC flow (G) shares the bottleneck with a TCP flow (T)

- The figure clearly shows that at lower bandwidths GCC is not able to get a fair share.

Results

➤ Figure shows the dynamics of the GCC flow and a TCP flow when $b = 1000\text{kbps}$ and a TCP flow is started before the GCC flow.

1. When the GCC flow joins the bottleneck it quickly gets a fair share until when, at around $t = 145\text{s}$
2. many consecutive REMB messages are received by the sender and the sending rate is decreased according to the value of A_r

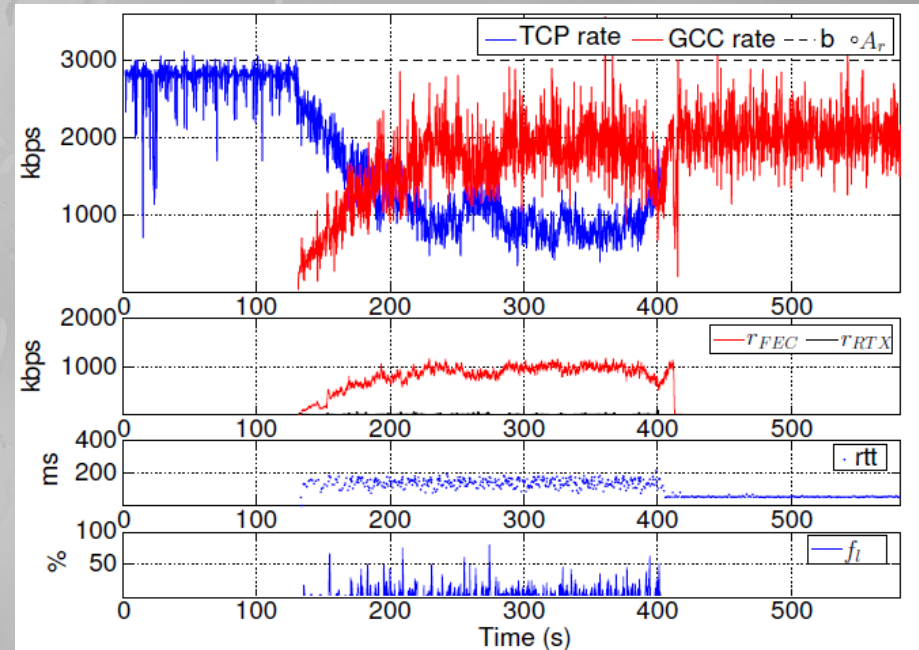


(a) $b = 1000\text{kbps}$, $RTT_m = 50\text{ms}$

Results

➤ Figure shows the results obtained when $b = 3000\text{kbps}$

1. the GCC sending bitrate is driven only by the sender-side controller
2. get a larger bandwidth than the TCP flow.



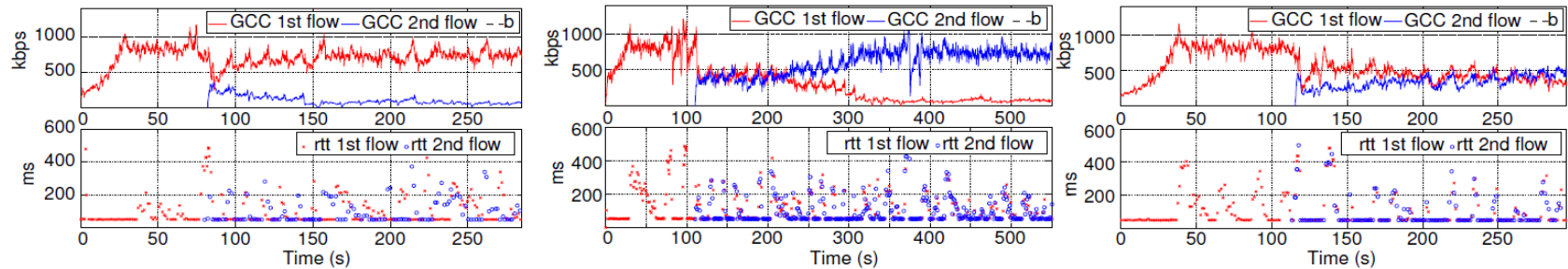
(b) $b = 3000\text{kbps}$, $RTT_m = 50\text{ms}$

Results

- Two concurrent GCC flows
- Assess the fairness of two GCC flows when sharing a bottleneck with constant capacity.
- $b_i \in \{1000, 2000, 3000\} \text{ kbps}$
- Propagation delay $RTT_m = 50 \text{ ms}$
- For each b_i run 5 tests.

Results

- It shows that the two GCC flows exhibit an unpredictable behavior



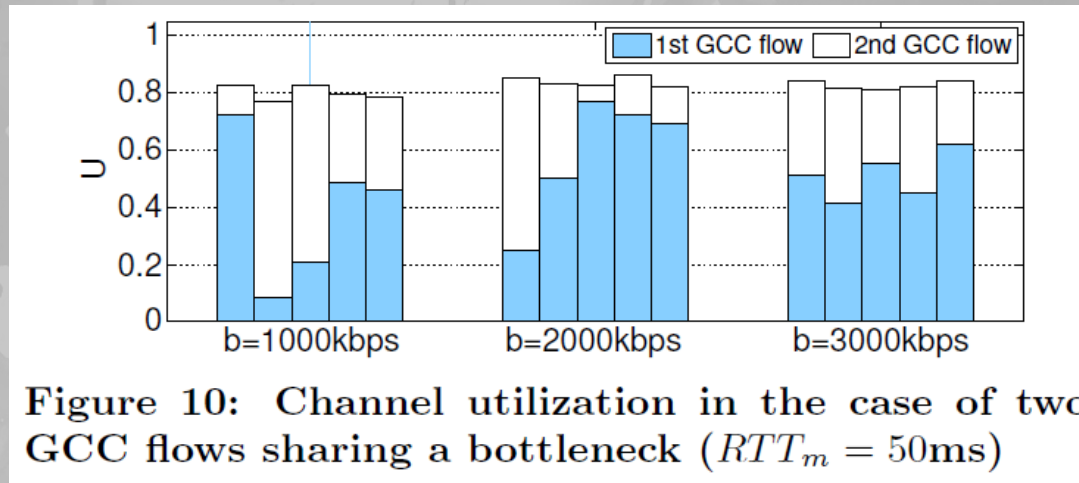
(a) The 2nd flow does not get a fair share (b) The 2nd flow starves the 1st flow (c) The flows share the bandwidth fairly

Figure 9: Two GCC flows share a bottleneck with $b = 1000\text{kbps}$ and $RTT_m = 50\text{ms}$

- a) the second flow obtains only a 10% channel utilization
- b) the first one that only gets 21% of the channel bandwidth
- c) the two flows fairly share the bandwidth.

Results

- Figure shows the channel utilization grouped for each of the considered available bandwidths b_i .



- in the case of $b = 1000kbps$ and $b = 2000kbps$ a very poor fairness is obtained
- $b = 3000kbps$, the two flow can roughly share the bandwidth fairness. And the first flow always gets a higher bandwidth share.

目 录

Background

Google Congestion Control

Experimental Testbed

Results


Conclusions

Conclusions

1. When a single GCC flow accesses the bottleneck, the channel utilization is above 0.8, even when the available bandwidth changes following a stair-case pattern, and queuing delays are contained
2. A video flow controlled by the GCC gets starved when sharing the bottleneck with a TCP flow if the bottleneck capacity is less or equal to 1000 kbps;
3. When two GCC video flows share the bottleneck, the algorithm behavior appears unpredictable and exhibit poor fairness.

Future Work

- The cause of the unfairness issues exhibited by GCC when sharing the bottleneck with TCP, or with another GCC flow, is currently under investigation.

The background of the entire image is a close-up of a glass surface covered in raindrops. Streaks of water run down the glass, and several out-of-focus light sources (bokeh) are visible, appearing as soft, glowing circles in shades of yellow and white. A semi-transparent dark grey rectangular box is positioned in the upper right area, containing the text.

谢谢观看