# New Results on RTP Scalability

**Jonathan Rosenberg and Henning Schulzrinne**
**Audio Video Transport wg**

# Overview

- Recap of Problem
- Recap of Reconsideration
- New Results
    - Linear Joins
    - Premature Timeouts
    - Reverse Reconsideration

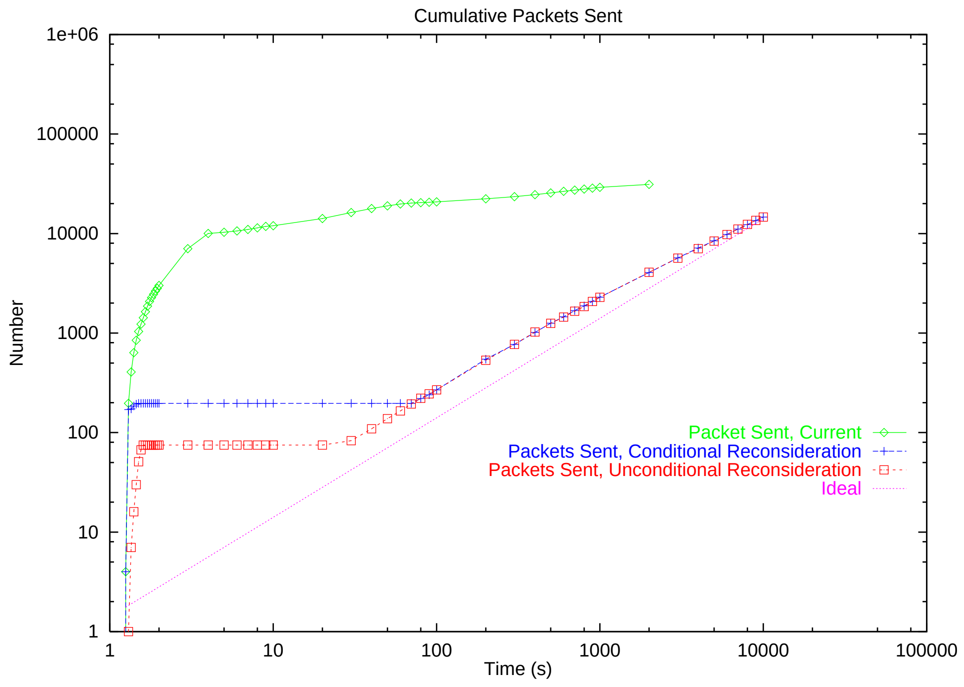
# Recap of Problem

- Will see near-simultaneous joins in many multicast RTP applications
- Transmission of RTCP packets by a host is based on observed group size
- Initially, each participant observes only one member - themselves
- Each transmits packets based on there being only one member - flood of packets and network congestion

# Reconsideration Algorithm

- When its time to send an RTCP packet, check group size.
  - Conditional: If group size has grown, recompute interval, check if result is before current time
  - Unconditional: Always recompute interval

# Results

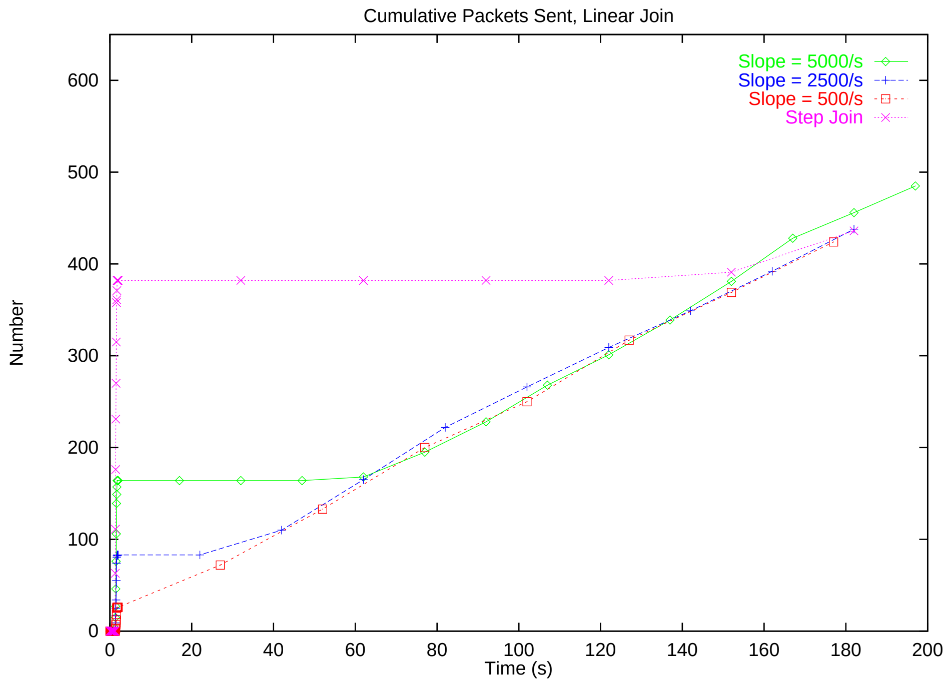


Cumulative Packets Sent
1e+06
100000
10000
1000
100
10
1
Number
1
10
100
1000
10000
100000
Time (s)
Packet Sent, Current
Packets Sent, Conditional Reconsideration
Packets Sent, Unconditional Reconsideration
Ideal

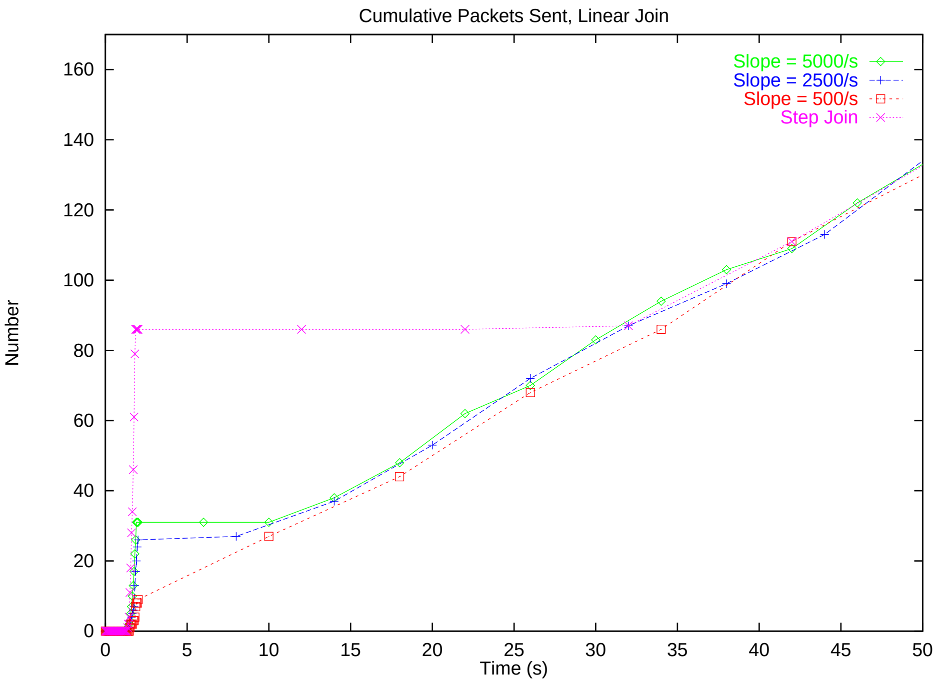
# Linear Joins

- Last time: concerns about “plateau effect”
- Plateau effect disappears when joins are linear
- 2500 users join, network delays are 300ms

# Conditional



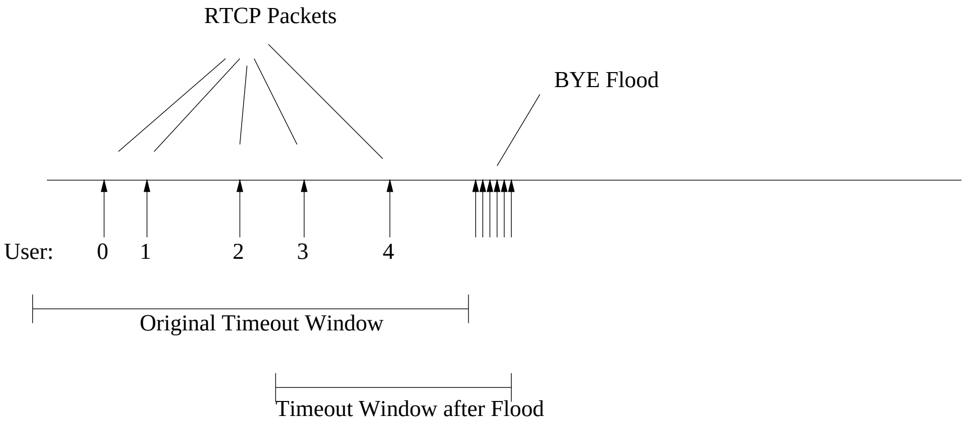
# Unconditional



# New Results

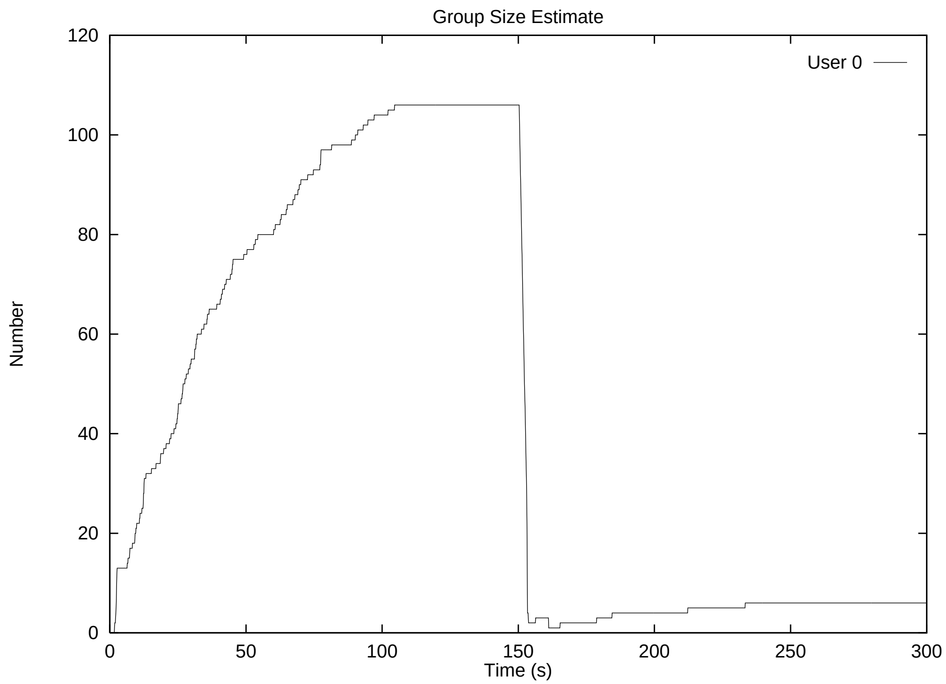
- Simultaneous leaves
- Causes a flood of BYE packets. Group size estimate quickly drops
- Timeouts based on current group size estimate, so timeout window decreases
- Current members haven't had a chance to send a new packet yet, so they time out

# Premature Timeouts



# Simulations

- **Classroom: teacher + 105 students join, 100 leave after class**



# Solutions

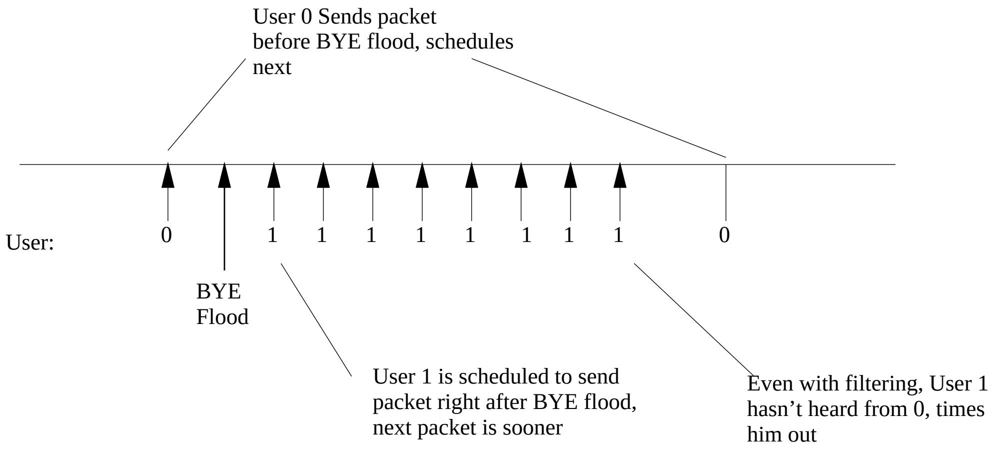
- **Effect observed by Sharma, Jacobson, et. al. in Infocom 97 - point to point case, this is** *distributed case*.
- **Two solutions:**
  - **Counting of rounds: look for refresh of a piece of "state", period between refreshes is** *round*, **timeout users who haven't sent in last $N$ rounds**
  - **Filtering: Apply exponential weighted average filter to state refresh rate, use average as the rate for computing timeouts**

# Solutions

- Counting of Rounds
  - Not appropriate for distributed scenario - concept of "round" where all state is refreshed is not well defined
  - Can lead to denial of service attacks: malicious user sends packets frequently, fools other receivers into thinking rounds are short - everyone times out
- Filtering
  - Seems reasonable
  - Must take into account distributed nature of system!

# Reverse Reconsideration

- System is distributed
- RTP end systems only adjust timers after transmitting (or reconsidering) a packet
- Result is delay between BYES and an increase in transmission rate



# Reverse Reconsideration

- **Solution: Reverse Reconsideration**
  - **Group size decreases at time $t_c$. Previous RTCP packet sent at time $t_p$, next is currently scheduled for $t_n$. Group size was $n_p$ when scheduled, now is $n_c$**
  - **At $t_c$, recompute:**

$$t_n = t_c + \frac{n_c}{n_p}(t_n - t_c) \tag{1}$$

$$t_p = t_c - \frac{n_c}{n_p}(t_c - t_p) \tag{2}$$

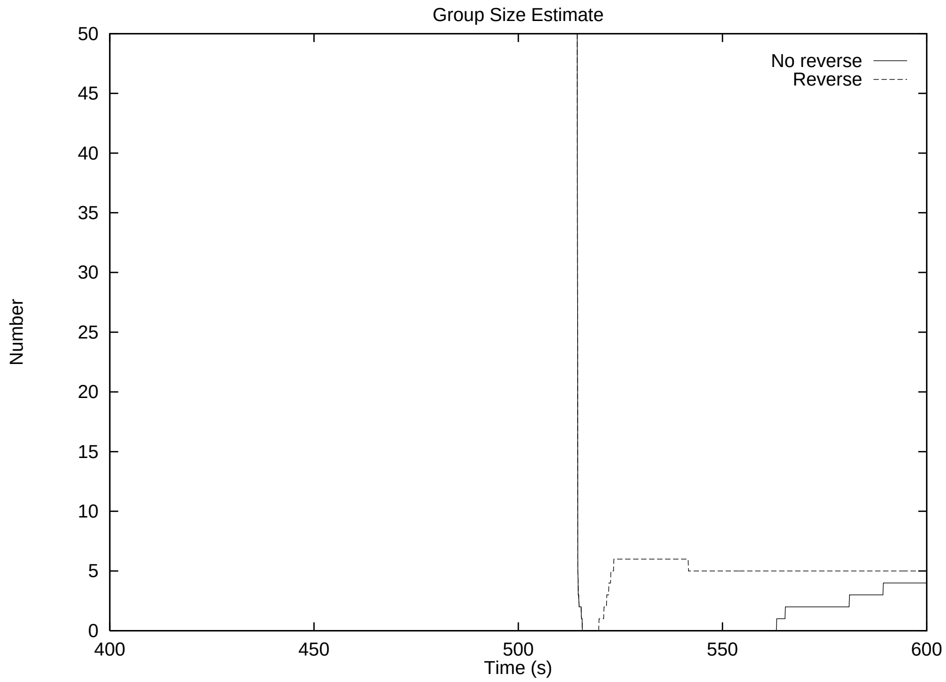
$$\tag{3}$$

# Reverse Reconsideration

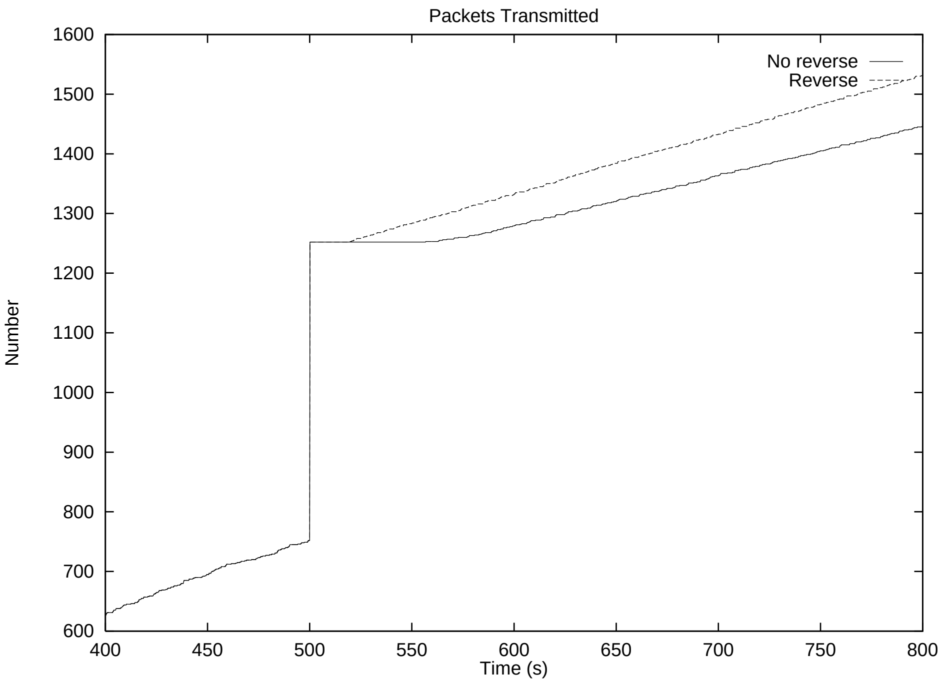
- Makes transmission rates adapt faster
- Safe - can only overestimate group sizes (normal reconsideration is more aggressive - its dangerous to underestimate group sizes)
- Keeps RTCP transmission rate at desired 5% of RTP rate

# Simulations

- 506 users join at $t = 0$, 500 leave at $t = 500$



# Simulations



# Conclusion

- Reverse Reconsideration looks good
- More work is needed to develop filter - what kind of filter, what parameters
- We don't expect interactions between "forward" and reverse reconsideration - must check to be sure
- Need to test on a real network - algoritms are being built into a C RTP library
- Developing a tool for automatically triggering near simultaneous joins and leaves in an RTP session