Low Latency Transport Mechanism

Behnam Montazeri,
Mohammad Alizadeh, John Ousterhout

May 28, 2015

Motivation

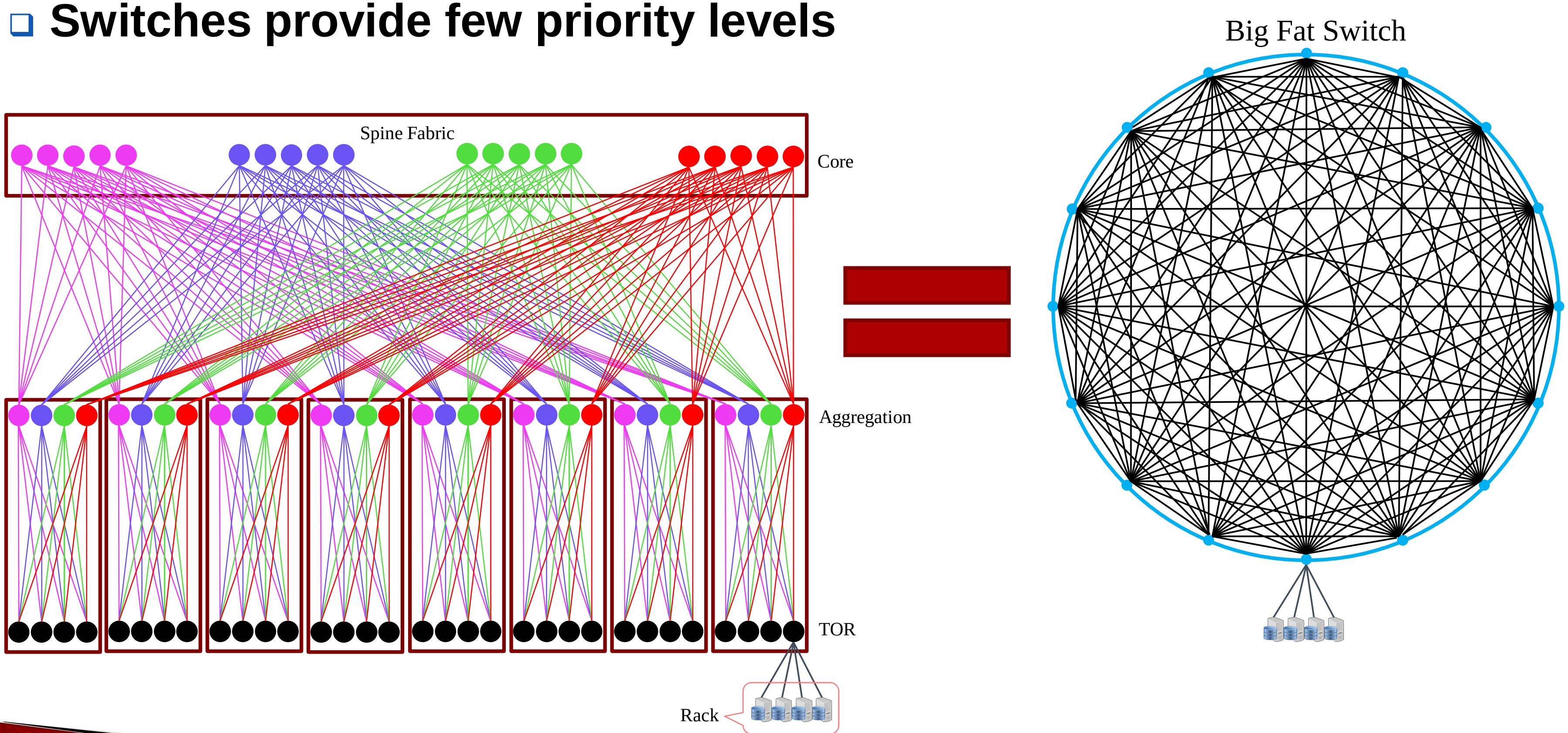
- **RAMCloud RPC currently uses Infiniband reliable transport**
- **Infiniband has scalability issues and is not commodity**
- **Ideally, we want low latency over unreliable datagrams**
- **Goal: To design a new reliable transport protocol**
  - ✓ Fitted for datacenter networks
  - ✓ Tailored for RPC systems

Objectives

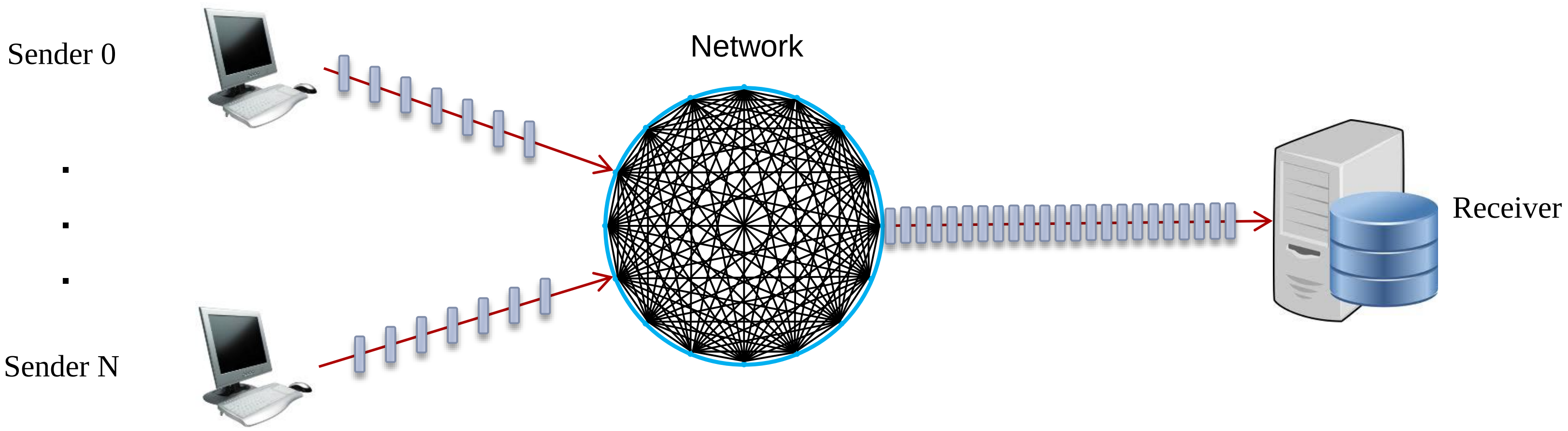
- **Low Latency**
  - As close as possible to hardware limit
  - Minimal Buffer Usage
- **Scalability**
  - One million client connections per server
  - Minimal per client state
- **Congestion Control**
  - Low latency for small request in presence of high network utilization

Network Assumptions

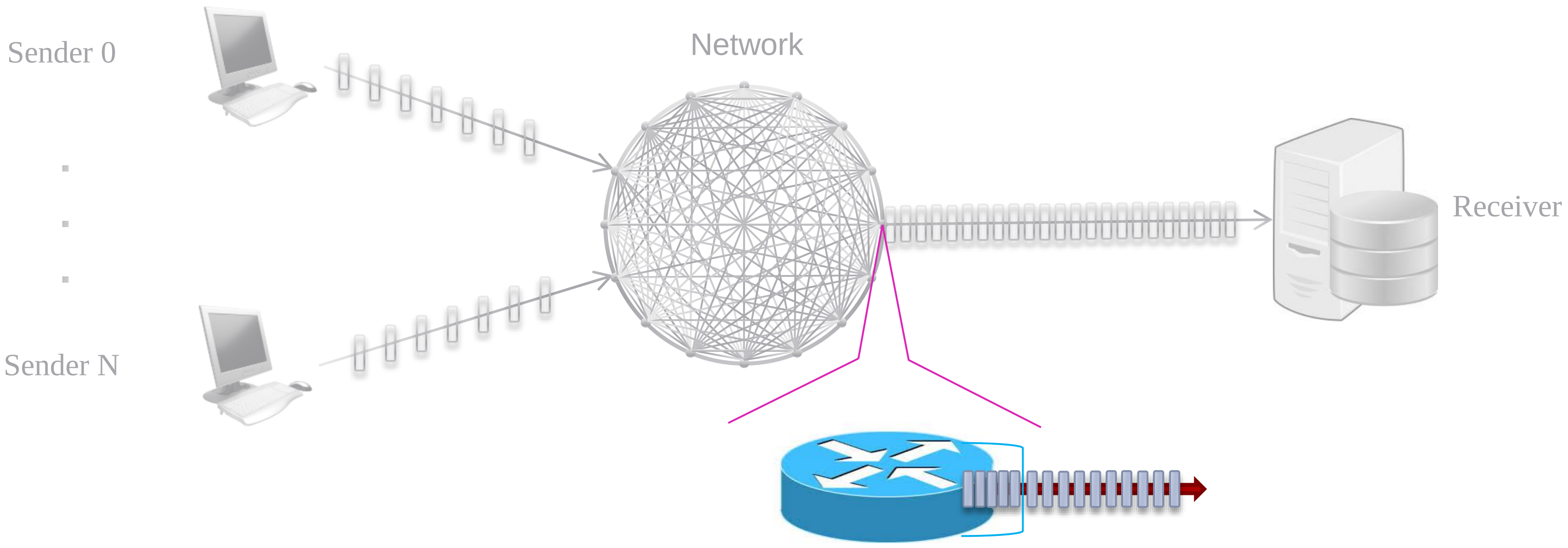
- Full Bisection Bandwidth Topology
- Load balanced
- Low latency Fabric
  - ✓ However, small random variations in latency are OK
- Switches provide few priority levels



Congestion Primarily at The Edge

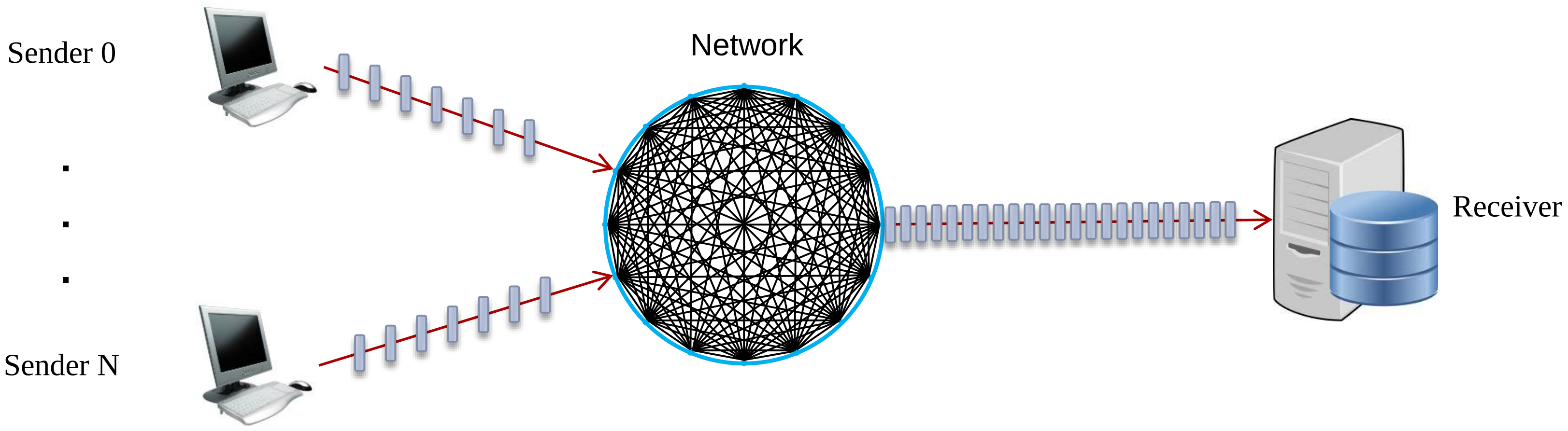


Congestion Primarily at The Edge



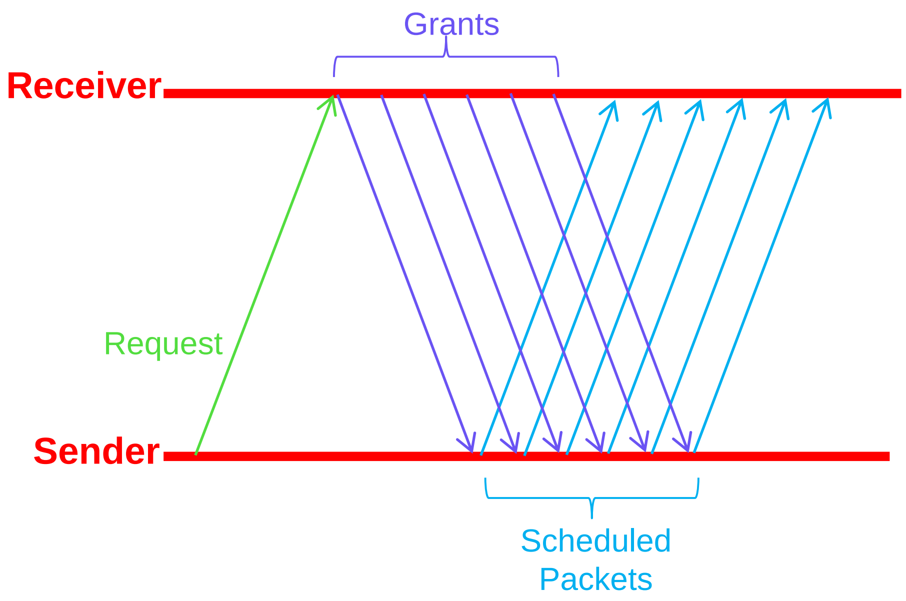
Congestion Primarily at The Edge

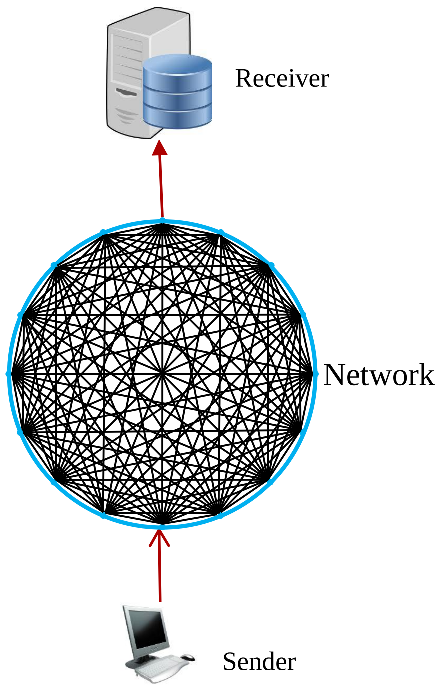
- Congestion primarily happens at the receiver's TOR
- Receiver has a lot of context to control congestion at the TOR
  - ✓ It sees all traffic through the edge link
  - ✓ It knows all message sizes



Receiver Side Congestion Control

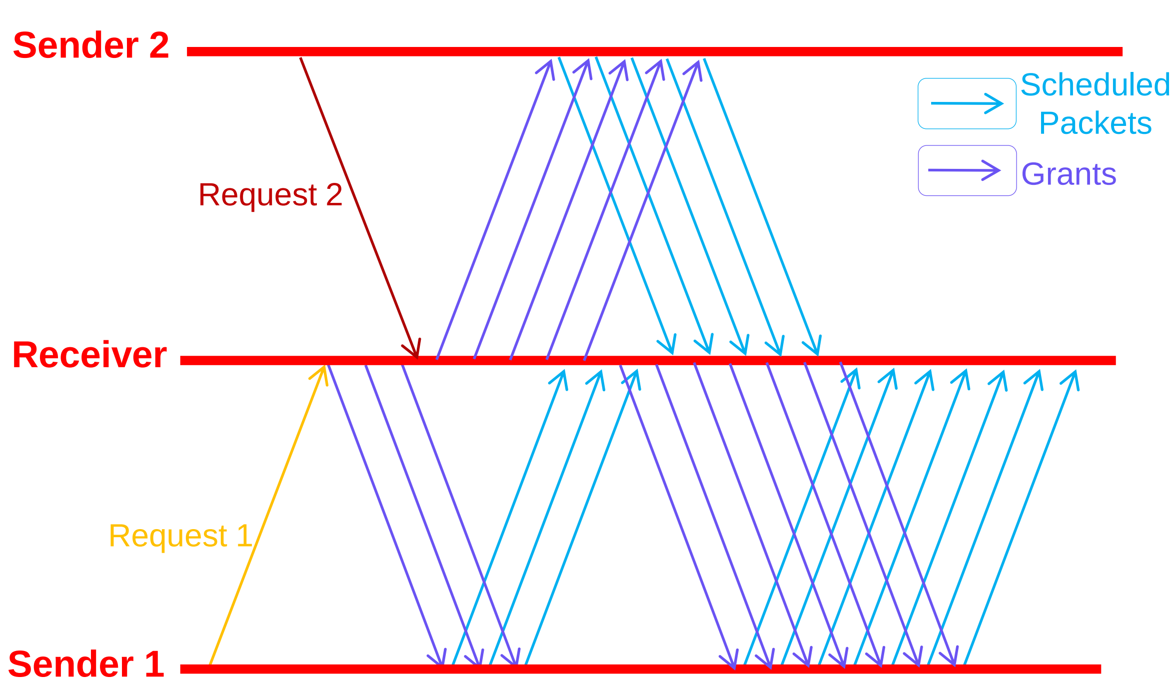
- **Example: One Sender, One Receiver**
- **Assume network delay is fixed**
- **Congestion Control Scheme**
  - ✓ Sender sends request that specifies the message size
  - ✓ Receiver sends grants (#of bytes permitted)
  - ✓ Grants are sent in fine grained time intervals (One packet time?)

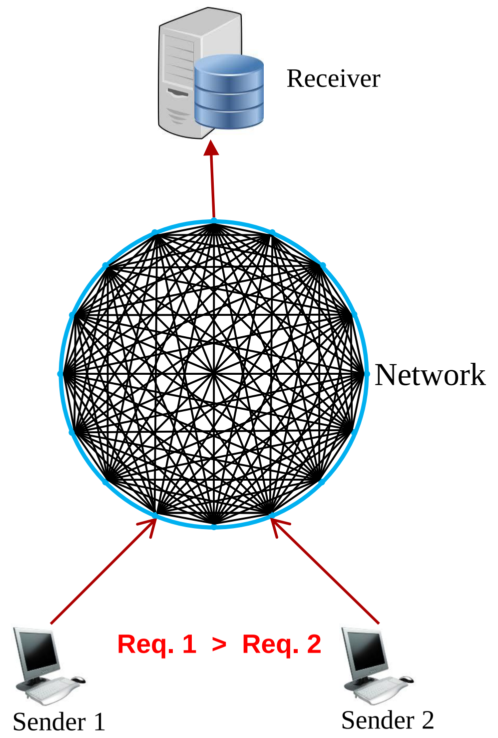




Using grants to preempt

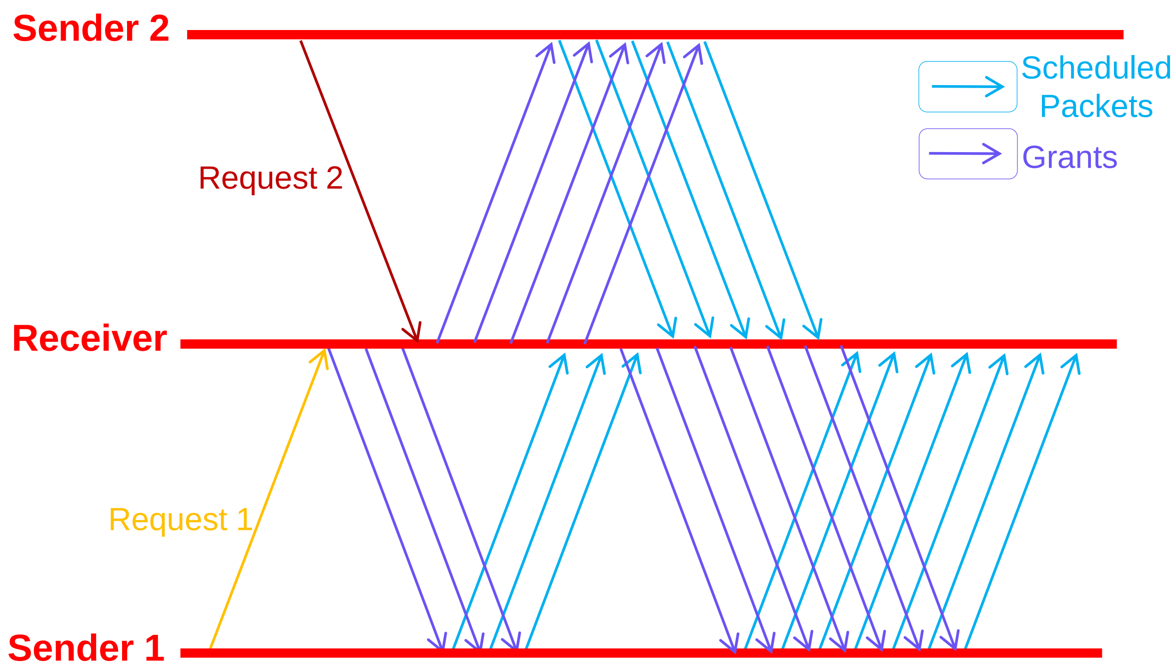
- **Multiple Senders**
  - ✓ Favor shortest request (Shortest Remaining Bytes First)
  - ✓ Use grants for preemption
- **Smaller grants => Faster preemption**

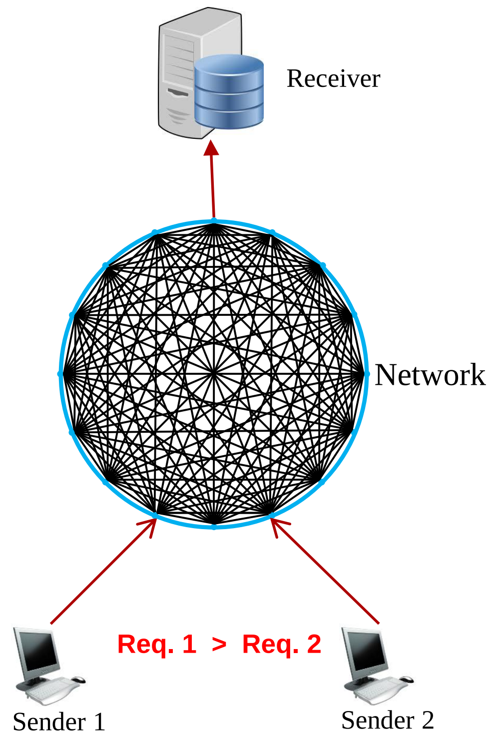




Using grants to preempt

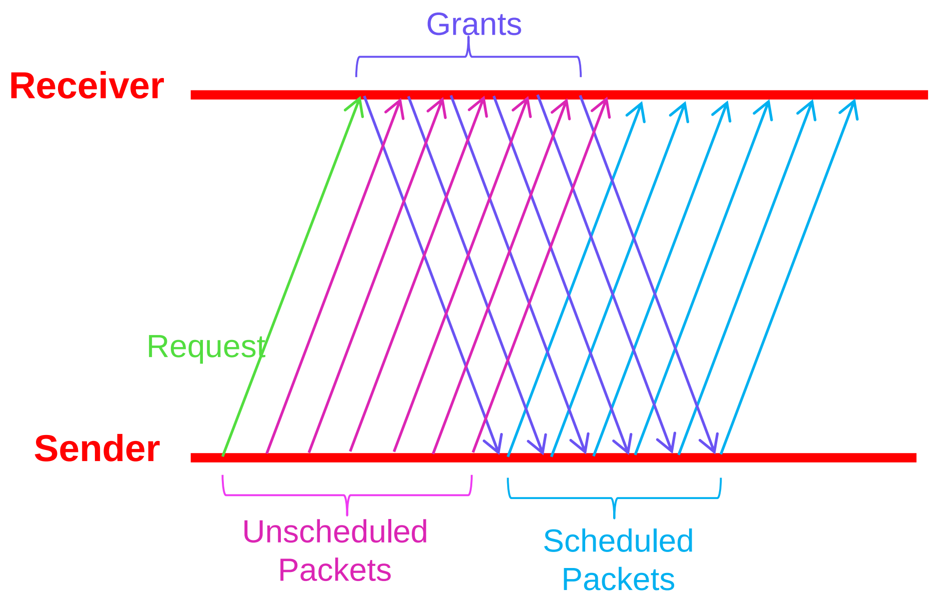
- **Multiple Senders**
  - ✓ Favor shortest request (Shortest Remaining Bytes First)
  - ✓ Use grants for preemption
- **Smaller grants => Faster preemption**
- **No buffering so far**





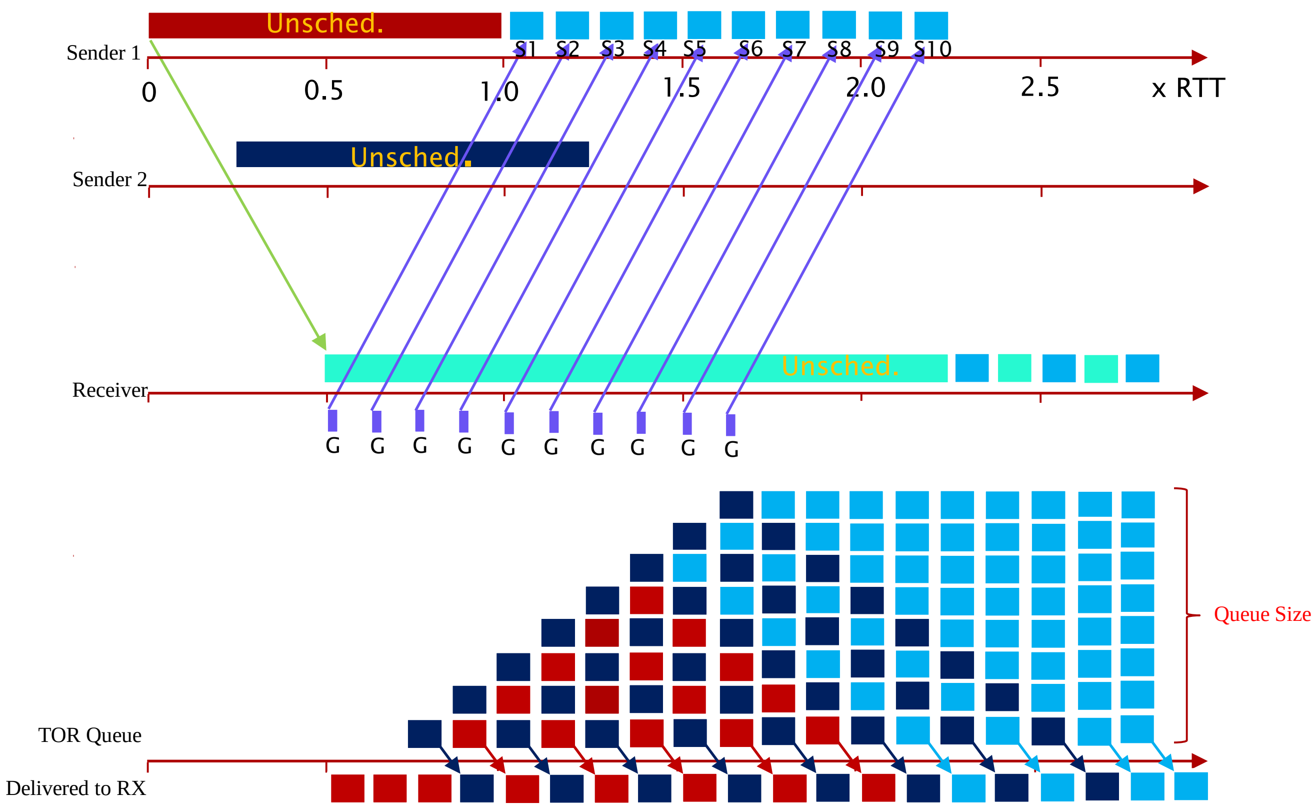
Unscheduled Traffic

- **Avoid extra RTT overhead**
  - ✓ Send a few *unscheduled packets*



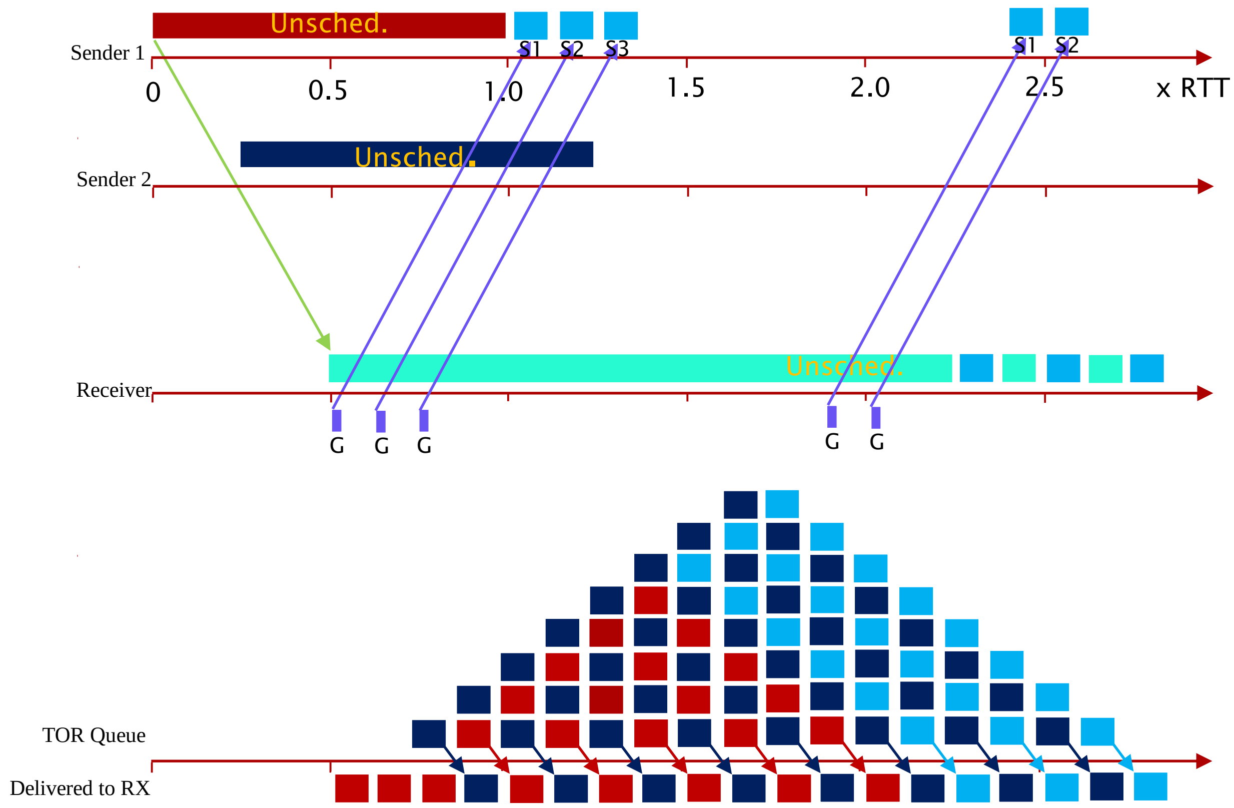
Problem: Buffer Build Up

- **Problems:**
  - ✓ With unscheduled traffic, multiple senders cause buffer build up
  - ✓ Buffering adds latency
  - ✓ Buffering limits our ability to preempt one request for another



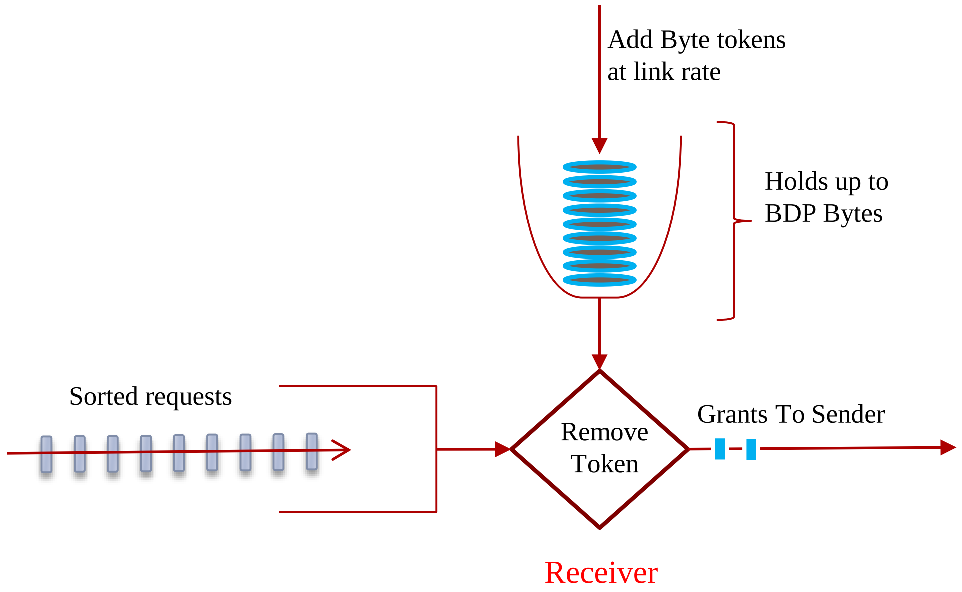
Problem: Buffer Build Up

- **Desired behavior:**
  - ✓ Defer sending grants until TOR queue is depleted

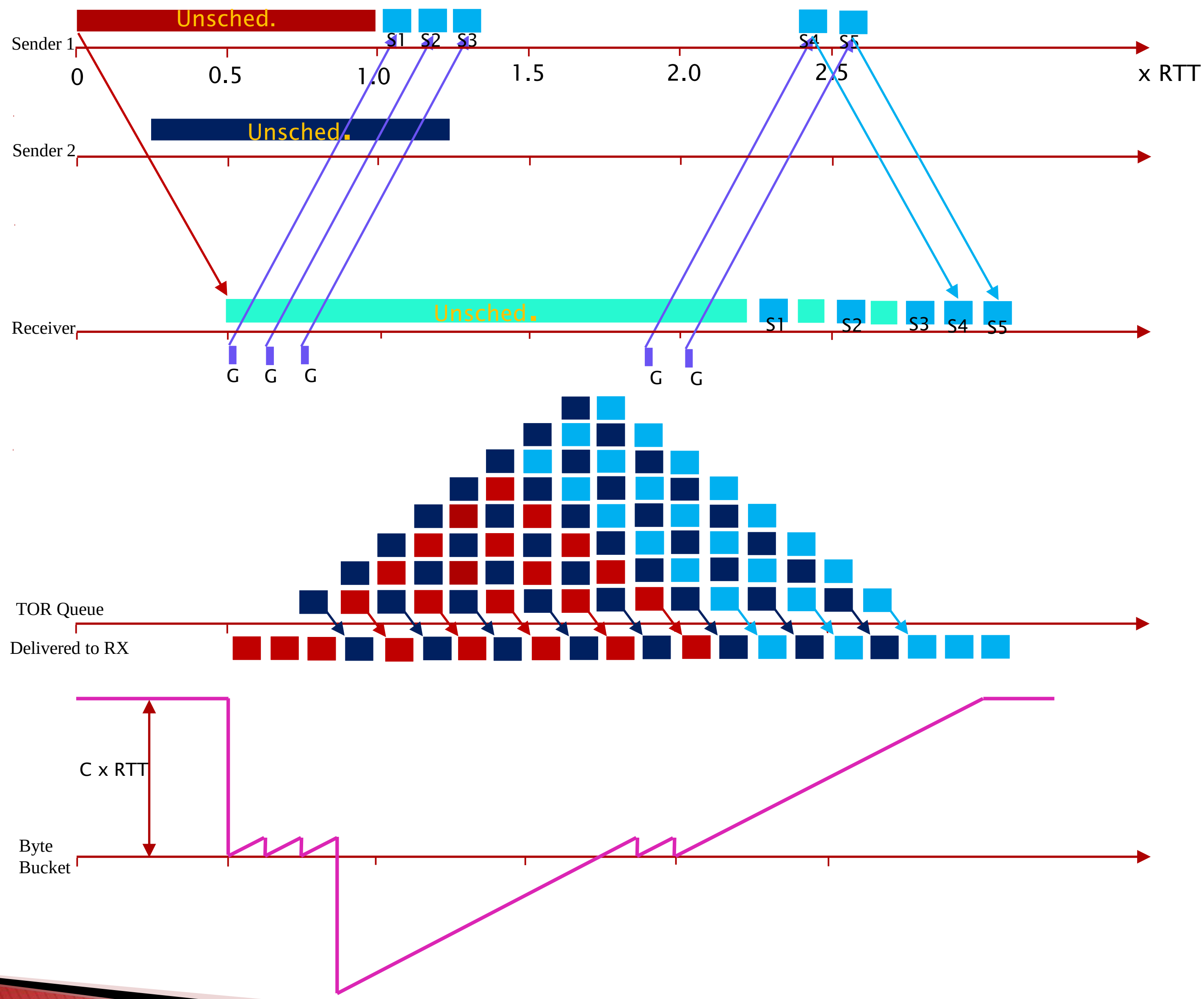


Solution: Byte Bucket

- **Byte Bucket**
  - ✓ Bytes are added to the bucket at link rate
  - ✓ Bucket level is capped at BDP = C x RTT
  - ✓ Unscheduled bytes are subtracted from bucket level



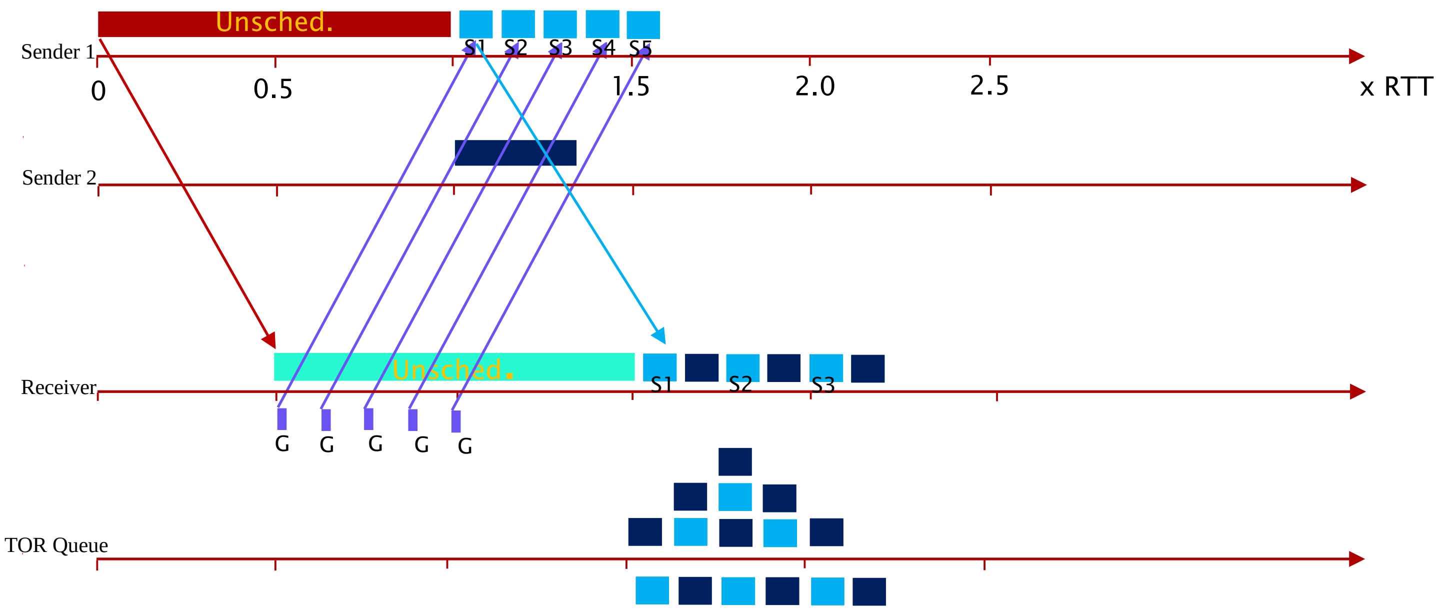
Solution: Byte Bucket



Preemption By Priorities

- **Allow preemption to favor short messages**
  - ✓ Utilizing small number of network priorities

- **Possible uses for priorities:**
  - ✓ Higher priorities for short messages



Preemption By Priorities

- **Allow preemption to favor short messages**
  - ✓ Utilizing small number of network priorities

- **Possible uses for priorities:**
  - ✓ Higher priorities for short messages
  - ✓ Higher priorities for unscheduled traffic
  - ✓ Utilizing multiple priorities within unscheduled traffic
  - ✓ Utilizing multiple priorities within scheduled traffic

- **Priorities are limited**
  - ✓ We should use a conservative approach in using them

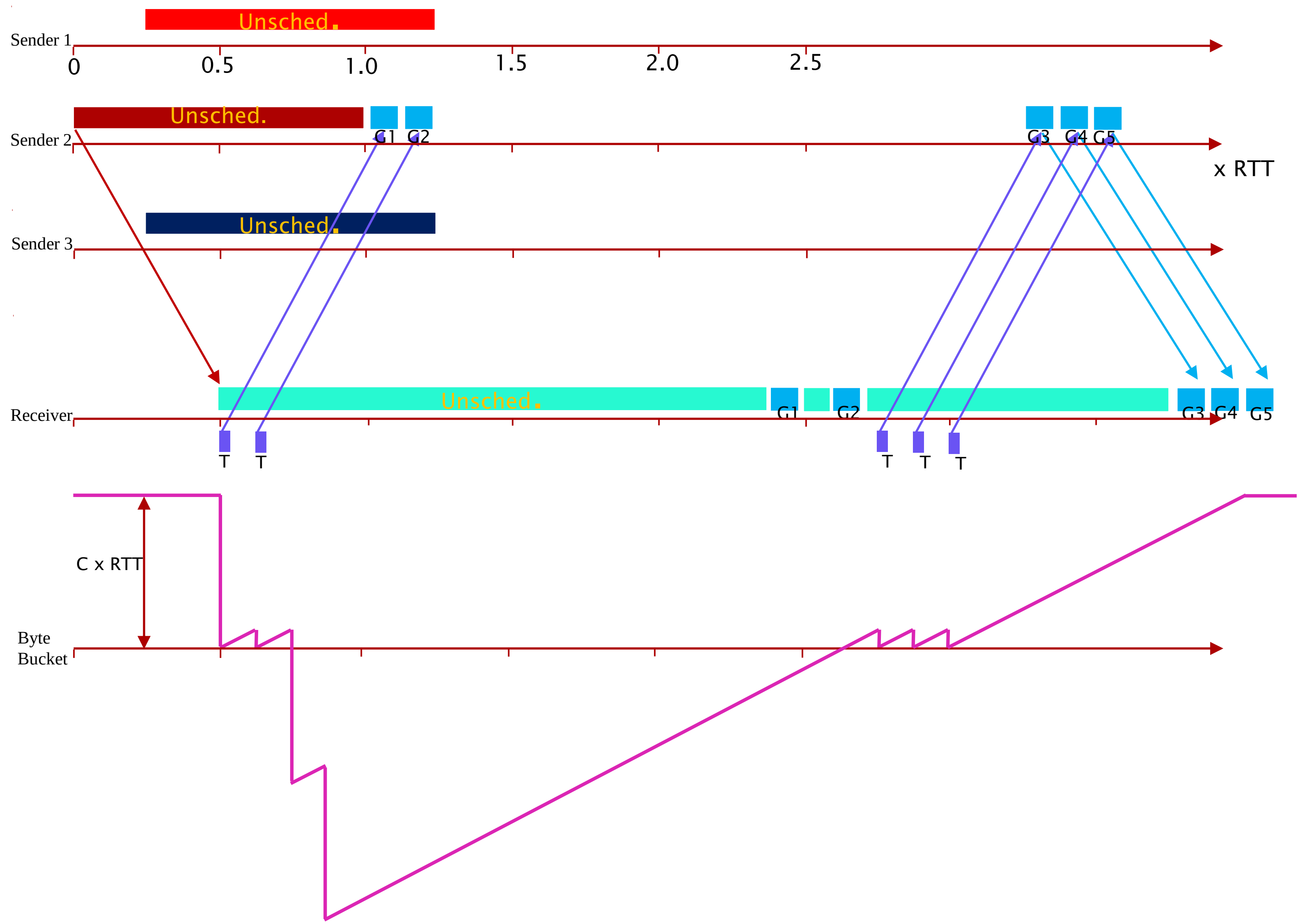
More problems

- **Uncontrolled unscheduled traffic**
  - Many senders send too much unscheduled traffic
  - Bucket level at receiver goes to large negative number
  - It will take a long time until bytes are accumulated in the bucket again
  - Receiver loses it's control over scheduling
- **Delay variations exists in network**
  - Packets take random path and pass through transient queues
  - Random variations cause bubbles to be blown up in the links
  - Idea: a little bit of queue can help with the bubbles
- **Senders have their own priority**
  - Two receiver send grant to a sender
  - A sender might want to prioritize receiver's one grant
  - Receiver's two grant is not efficiently used
- **Packet losses**

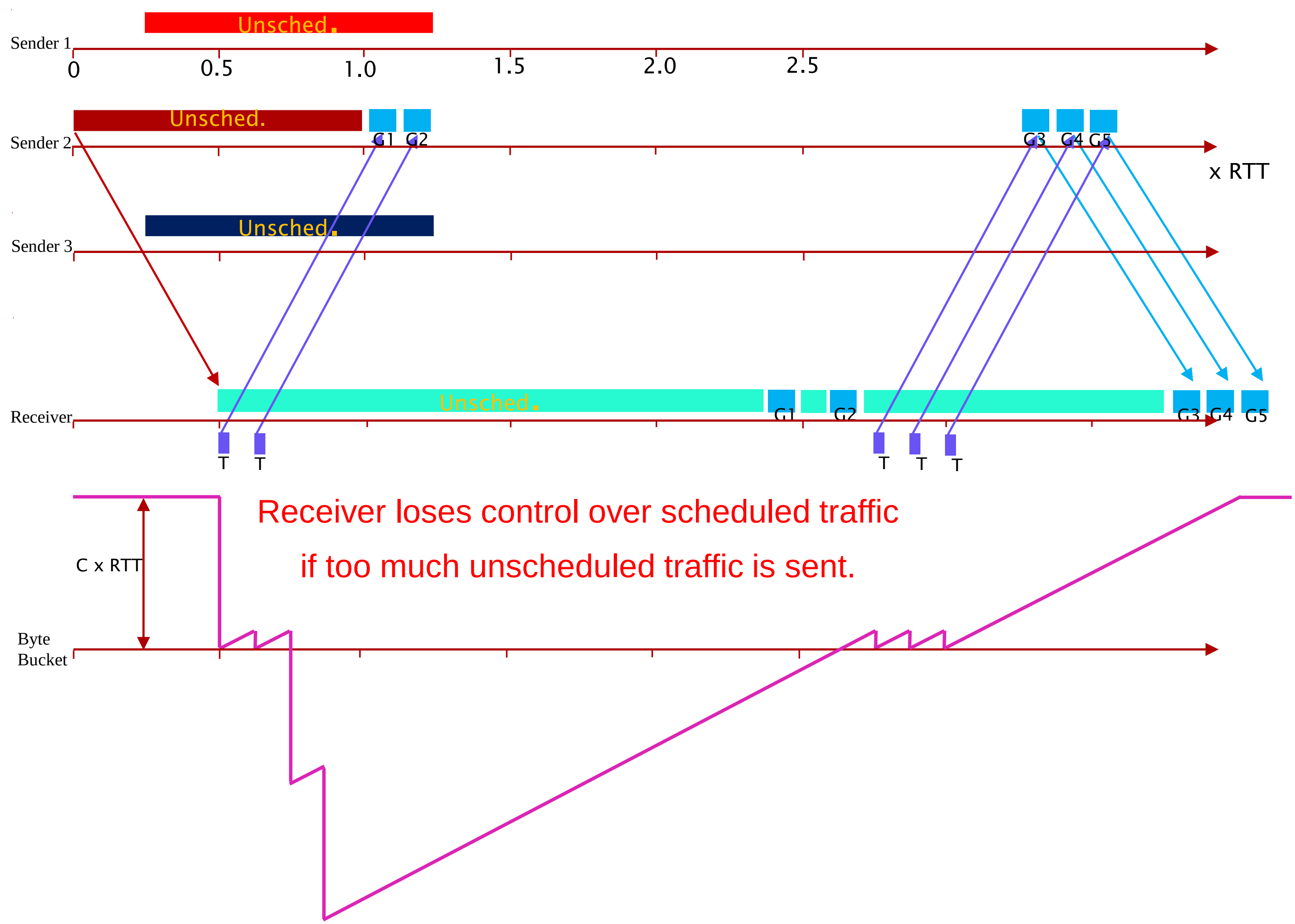
Conclusion

- **A new transport for RPC systems in datacenter networks**
  - ✓ Receiver side congestion control
  - ✓ Near ideal network latency and flow scheduling
- **Work in progress**
- **Current State**
  - ✓ OMNeT++ Simulations to characterize latency variations in DCN
  - ✓ Discussions about specifics of the algorithm (not complete)
- **Next steps**
  - ✓ More work to be done to flesh out the algorithm
  - ✓ Simulation of the algorithm in OMNeT++
  - ✓ Comparison of the algorithm performance to existing approaches
  - ✓ Implementation in RAMCloud RPC

Problem: Controlling Unscheduled Traffic



Problem: Controlling Unscheduled Traffic

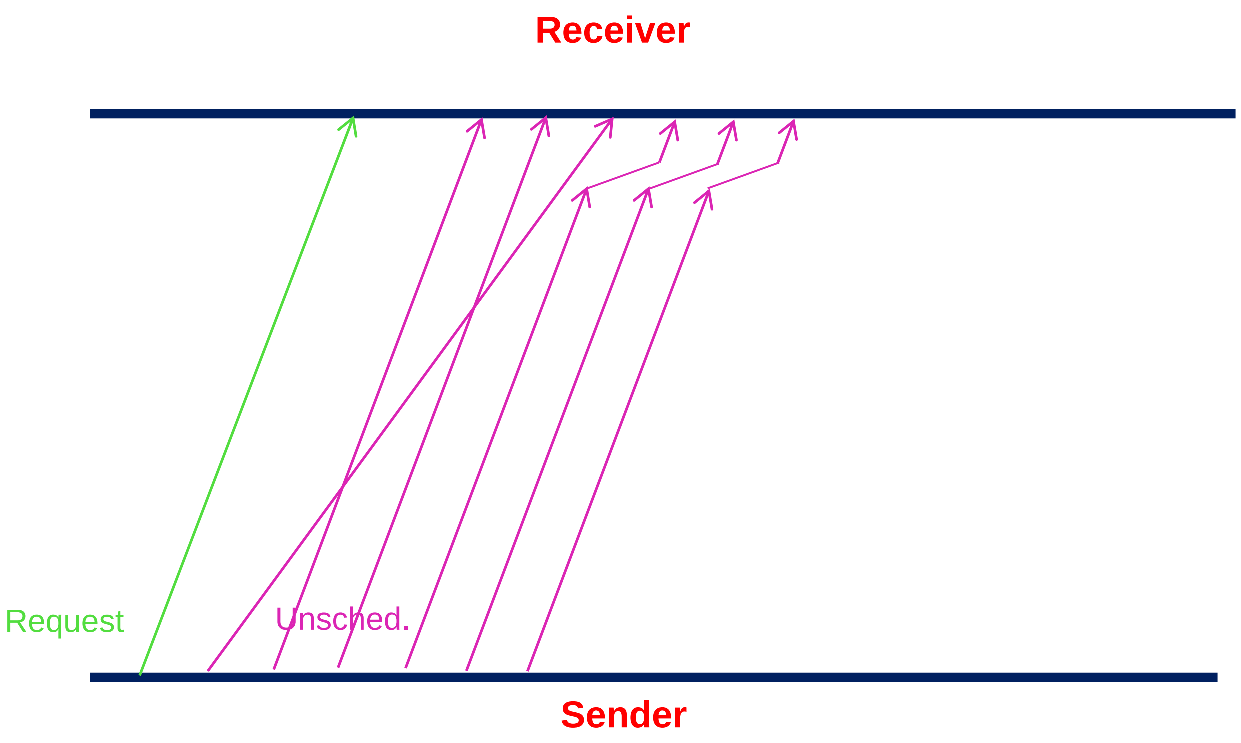


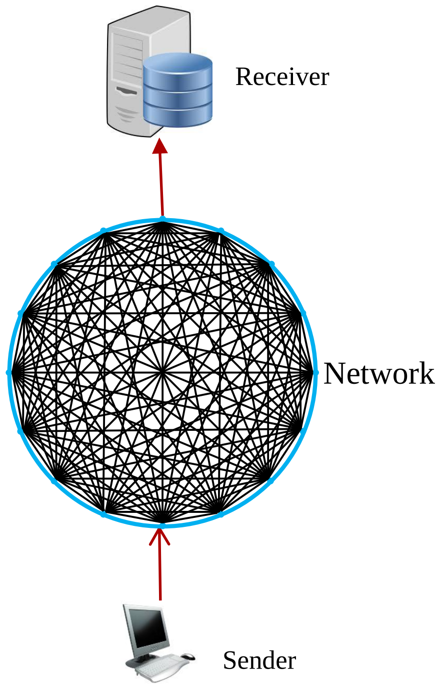
Receiver loses control over scheduled traffic

if too much unscheduled traffic is sent.

Problem: Network Delay Variations

- One Sender, One Receiver
- Network delays may have random variations





Problem: Network Delay Variations

- One Sender, One Receiver
- Network delays may have random variations
- Small amount buffering at TOR queue helps with the variations