Achieving Real-Time Communication over Ethernet with Adaptive Traffic Smoothing

By Seok-Kyu Kweon, Kang G. Shin and Gary Workman

Outline

- Introduction
- Problem Statement and Approach
- Experiment Evaluation on Linux
- Experiment Evaluation on Windows NT
- Conclusion

Introduction

- Ethernet becomes the preferable technology for real-time control network
- Unpredictable delay characteristic due to 1-persistent CSMA/CD protocol
  - Contention with non-RT packets in local node
  - Collision with packets from other nodes
- Solution: Traffic smoother
  - RT packets receives higher priority
  - Smooth non-RT streams to reduce collision

Problem Statement and Approach

- Scenario: automated manufacturing facility
- Traffic Pattern: event-driven RT message generation in pseudo periodic manner & bursty non-RT traffic

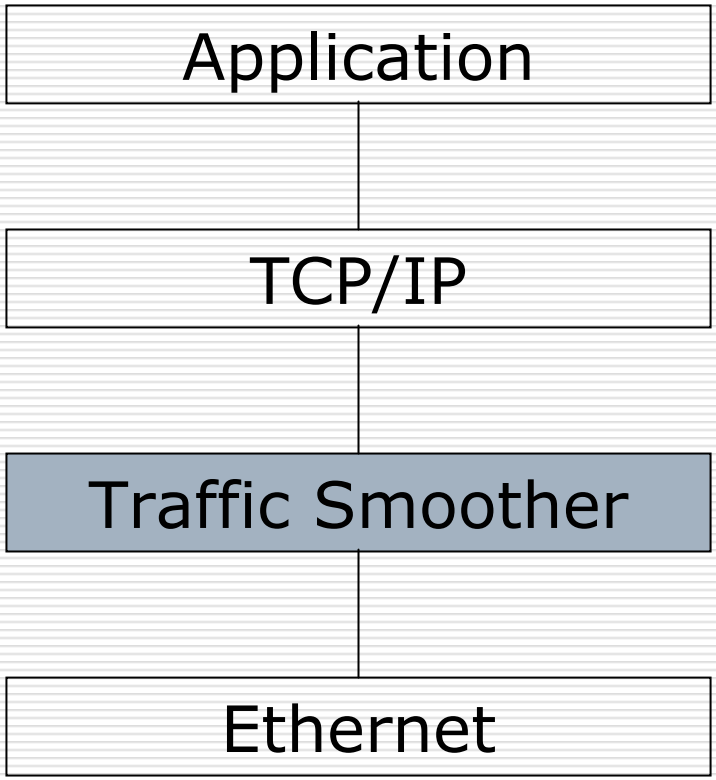
Fixed Rate Traffic Smoothing

- Regulate bursty packet streams
- Provide statistical bound of deadline-miss ratio by modeling CMSA/CD with Exponential Binary Backoff as semi-Markov process
  - Derive relation between delay and network utilization
- Enforce fixed station input limit to keep network utilization low to provide soft real-time communication

Smoother algorithm

- Introduce traffic smoother between TCP/IP layer and MAC layer
  - Distorted traffic pattern if implemented over TCP/IP layer (slow start….)
- Regulate packet stream using credit bucket (more like token bucket)
  - CBD: Credit Bucket Depth
  - RP: Refresh period

Smoother Architecture



Smoother algorithm(Cont')

- Up to CBD credits are added to bucket every RP seconds
- Overflow credits are discarded
- Smoother forwards the packets to MAC if at least one credit is available
- Decrement credit number as the size of packet
- Allow negative credit balance
- Example: Avg. throughput fixed at 312.5 KB/sec
  - (CBD, RP) = (1500, 0.0048)
  - (CBD, RP) = (150000, 0.48) → burstier output

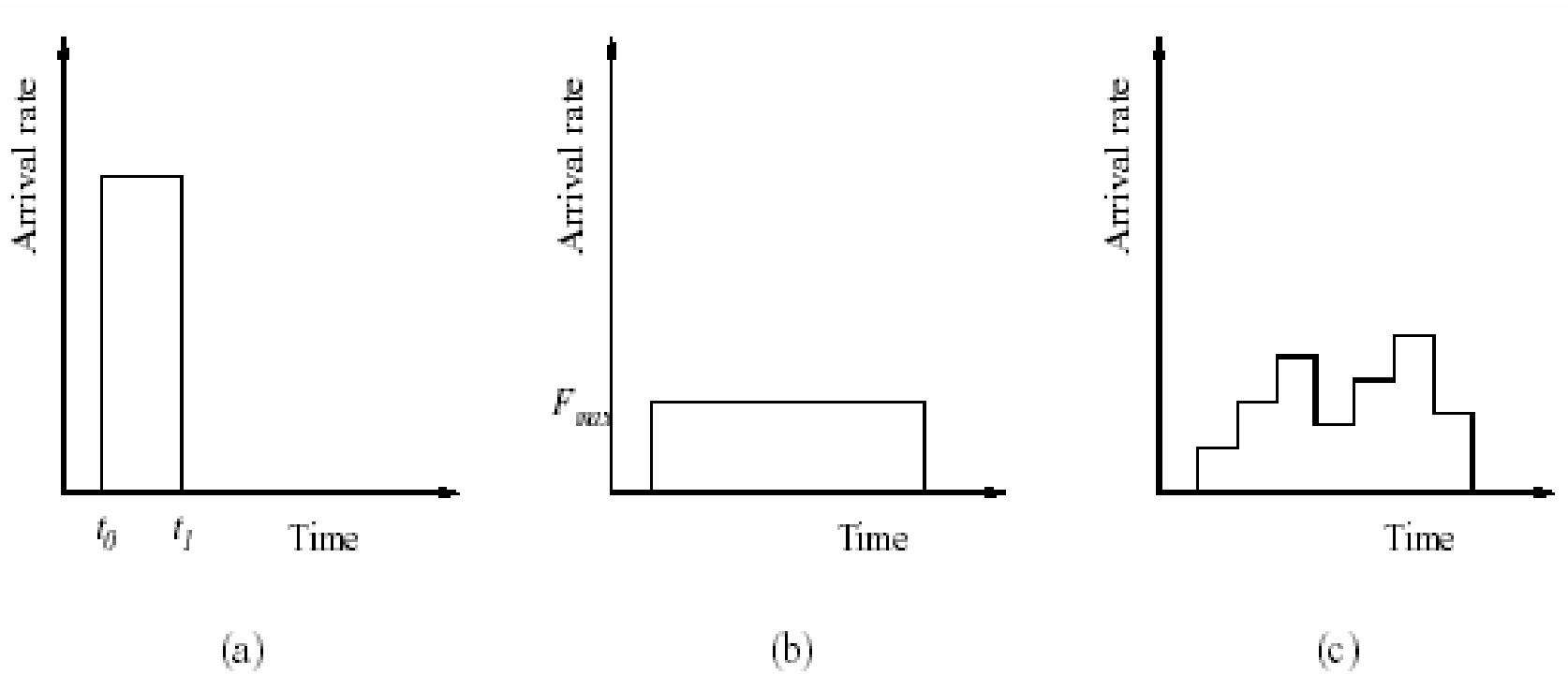
Smoother algorithm (Cont')

- Assign higher priority to RT packets
  - Delay non-RT packets to abide station input limit
  - Extra RT packets can further delay non-RT packets
  - RT traffic arrives pseudo-periodically and is already smoothed.

Problems

- Inflexible and unscalable
  - Station input limit is reduced as number of nodes increases in order to maintain fixed global input limit
  - Introduce large delays to non-RT packets
- Solution: Adaptive-Rate traffic smoothing
  - Allowing varying max traffic generation rate depend on network load
  - Non-RT traffic generation is allowed to increase
  - Modification to current protocol is minimal

Adaptive rate traffic smoothing



Original Traffic Arrival

Fixed-Rate Traffic Smoother

Adaptive-Rate Traffic Smoother

Issues

- How to detect network utilization
  - Indirect methods
    - packet collision or buffer clearing rate
    - Measure network utilization in promiscuous mode
- How to adapt to the change
  - Tuning throughput by changing CBD and RP
    - Changing CBD → fluctuated burst size
    - Changing RP → better choice

Harmonic Increase & Multiplicative Decrease Adaptation

- Similar to slow-start increase and multiplicative decrease concept
- Increase: "Harmonically" increase station input limit by decreasing RP by Δ (every τ time)
- Decrease: Detect Packet Collision
  - Clear all credits in bucket
  - Delay non-RT packets
  - Doubles RP

Procedure smoothing

If (Packet.TypeOfService = RealTime) then {

send_to_NIC;

CurrentNetworkShare:= CurrentNetworkShare – Packet.Size;}

Else if (LastCollisionTime ≥ CurrentTime - α) then {

send_back_to_queue;

CurrentNetworkShare := 0;

RP = min (RP_{max}, 2 × RP);}

Else if (CurrentNetworkShare > 0) then {

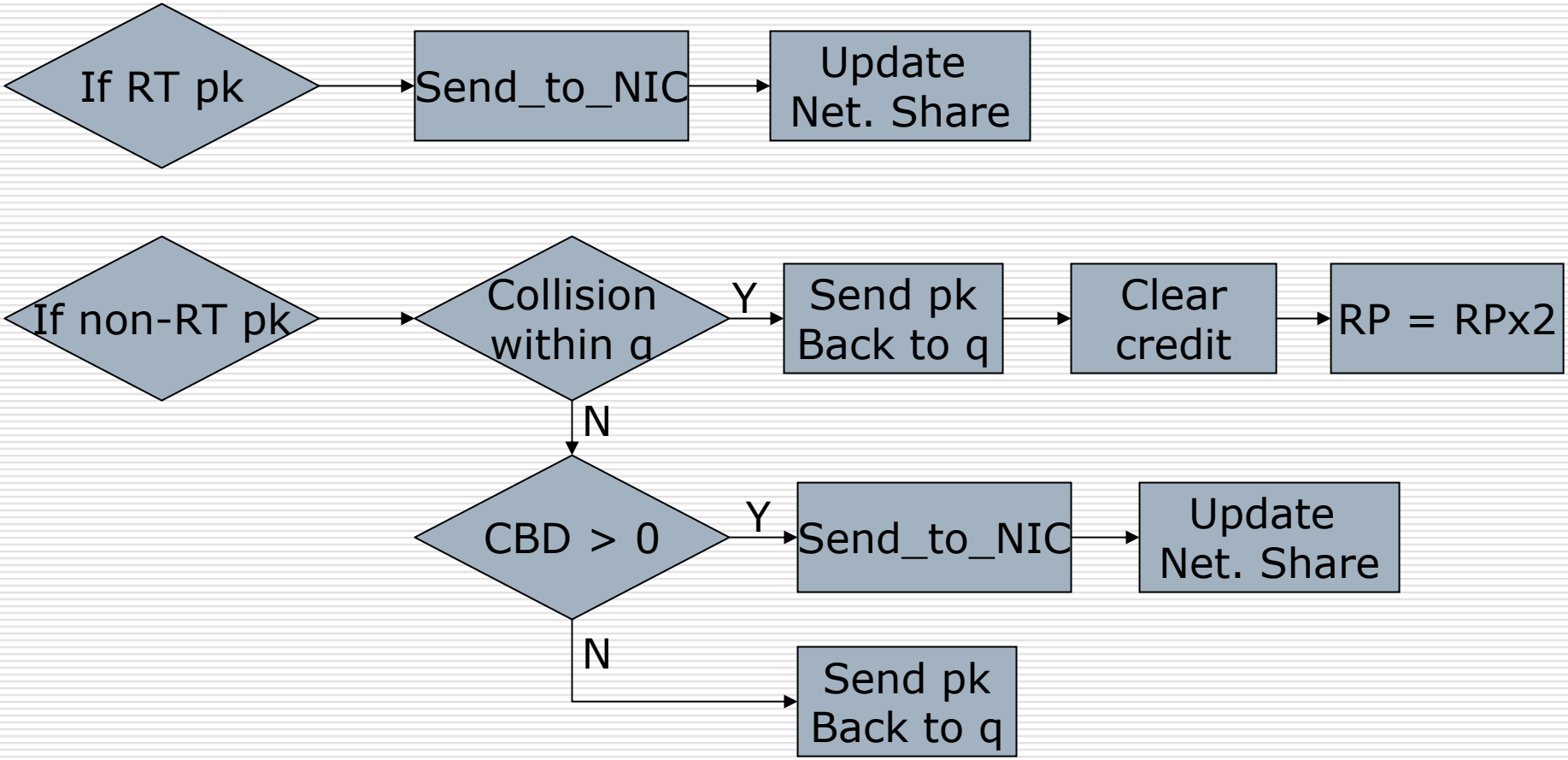
send_to_NIC;

CurrentNetworkShare := CurrentNetworkShare – Packet.Size;}

Else

send_back_to_queue;

Flowchart for smoothing



Procedure refresh

RP := max (RPmin, RP - Δ);

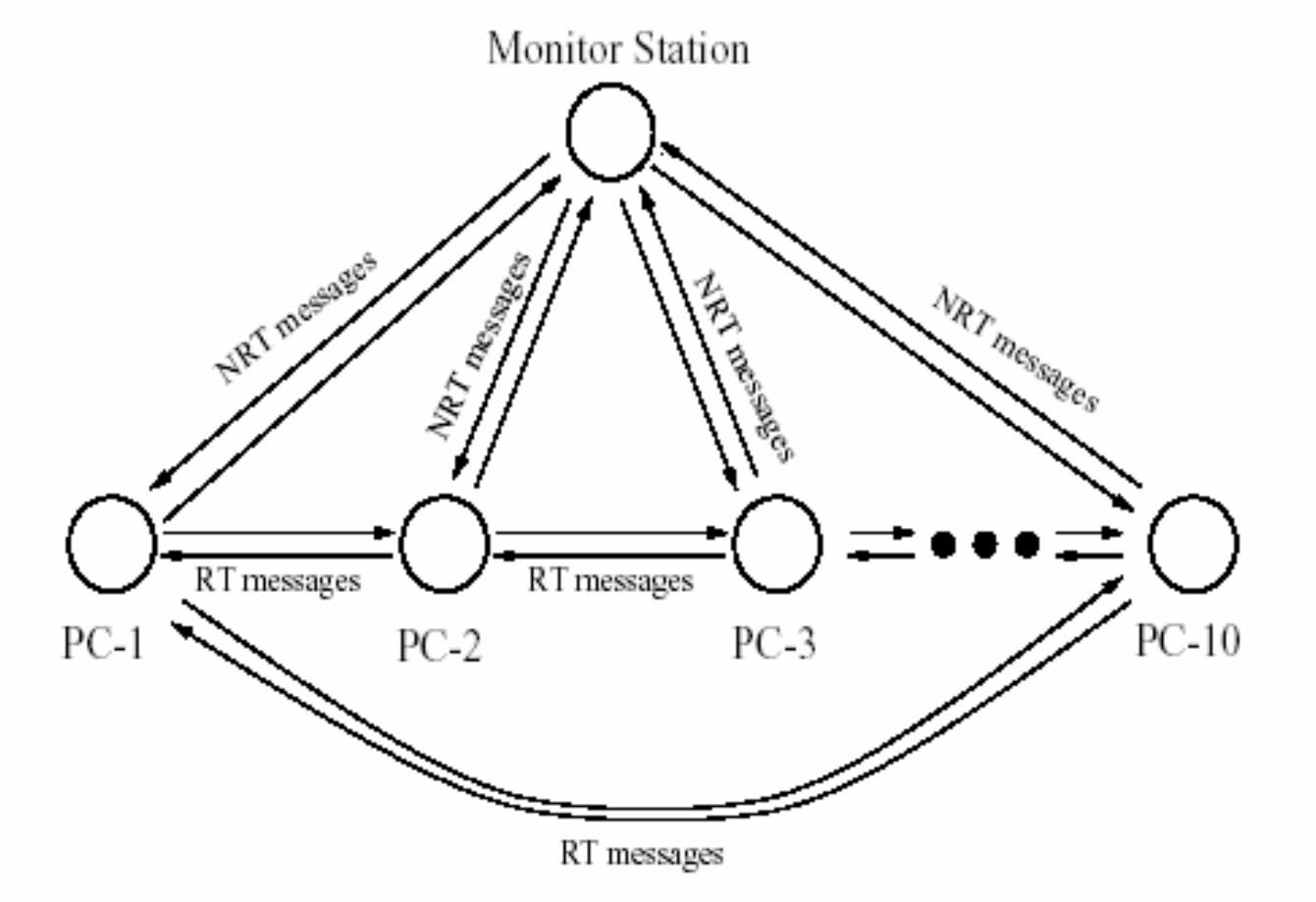
If (CurrentTime = NextRefreshTime) then {

CurrentNetworkShare := min(CurrentNetworkShare + CBD, CBD);

NextRefreshTime := CurrentTime + RP;

}

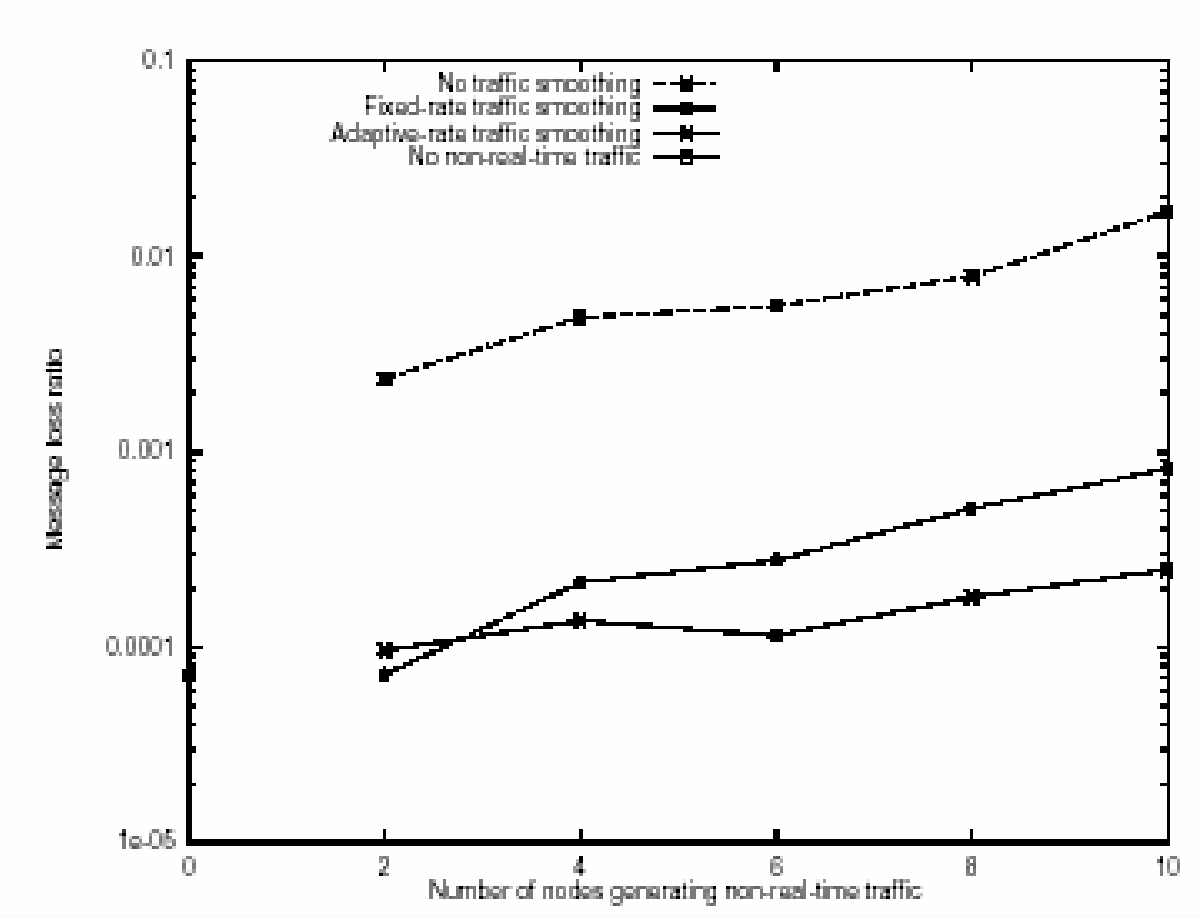
Experiment Evaluation on Linux



Environment Setup

- 1 Monitor Station, 10 PCs
- PC are arranged into a logical ring topology, exchanging RT control messages with neighbors
  - 100 byte in size
  - 0.3 sec inter-arrival time, exp. distributed
  - 2×100×8×10/0.3 = 53.3Kbps
- PC generates non-RT traffic when probed by monitor station
  - 1M byte in size
  - Non-greedy mode: 25 sec inter-arrival time, exp. distributed =>320Kbps/PC
  - Greedy mode: send non-RT traffic continuously

Experiment result: RT message loss ratio

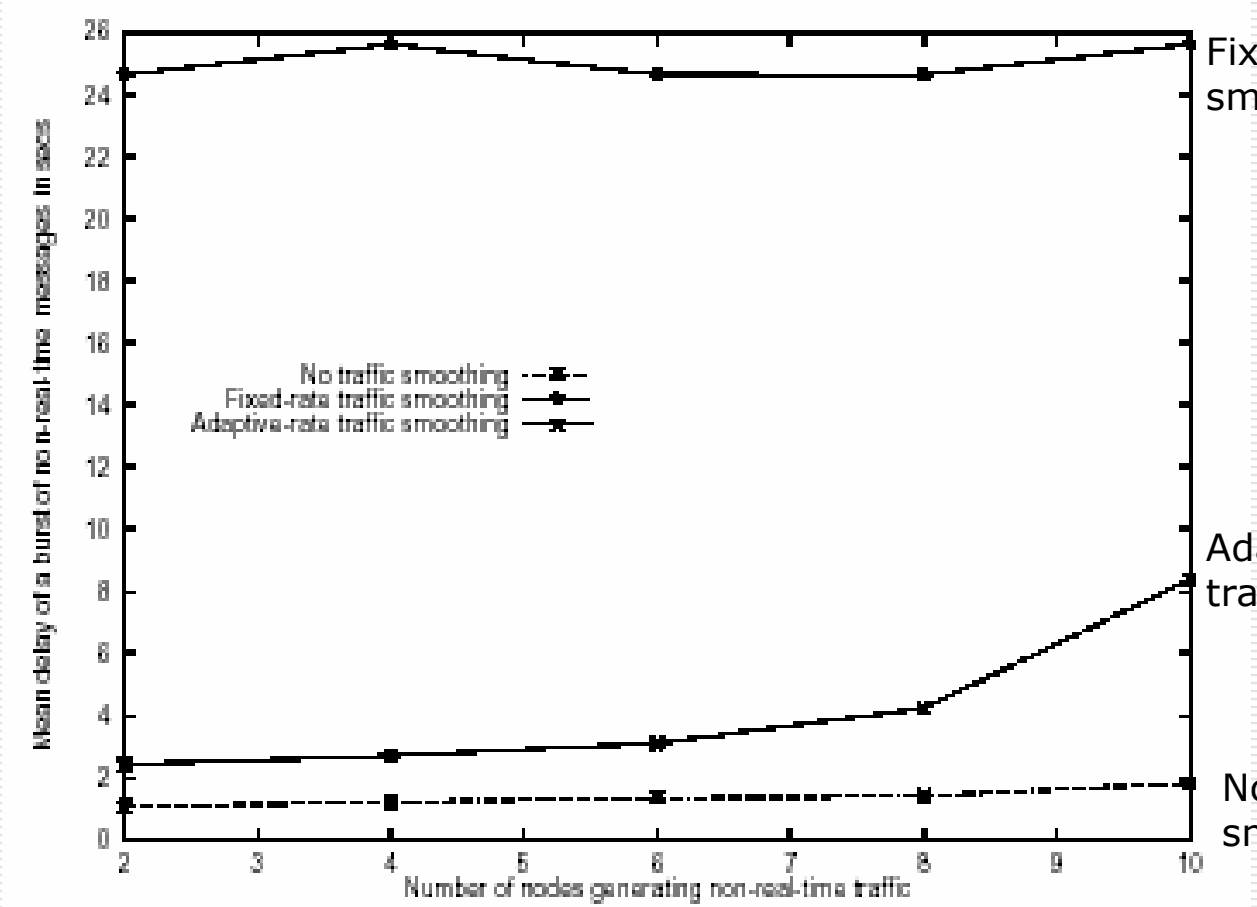


No traffic smoothing

Fixed-rate traffic smoothing

Adaptive-rate traffic smoothing

Experiment Result: mean delay for non-RT messages

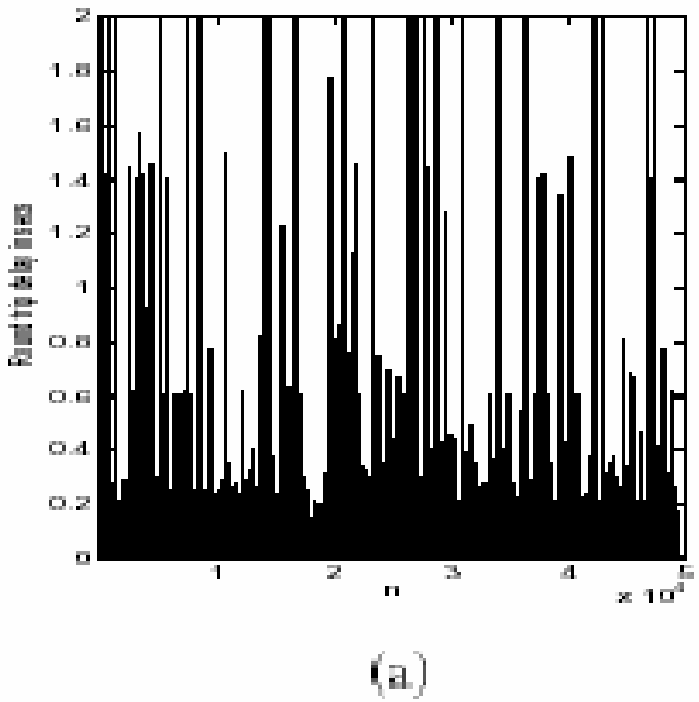


Fixed-rate traffic smoothing

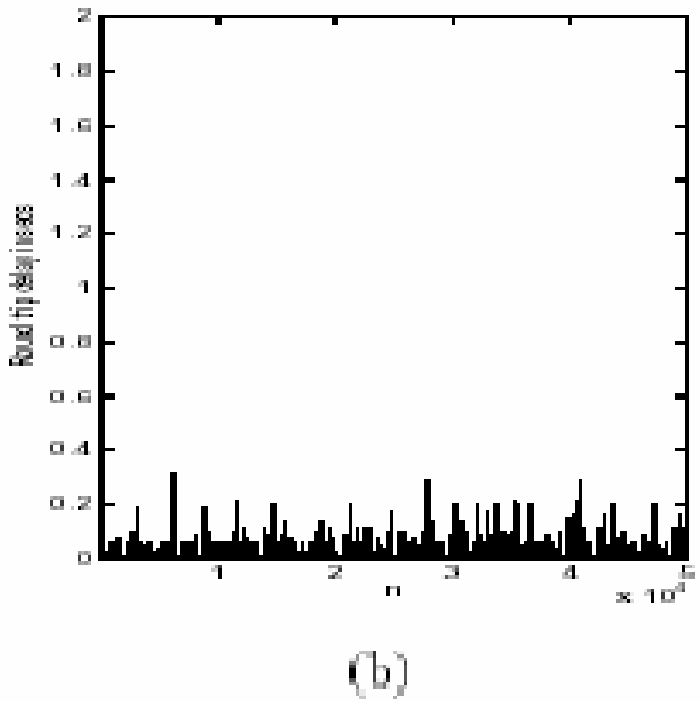
Adaptive-rate traffic smoothing

No traffic smoothing

Experiment Result: delay for RT messages

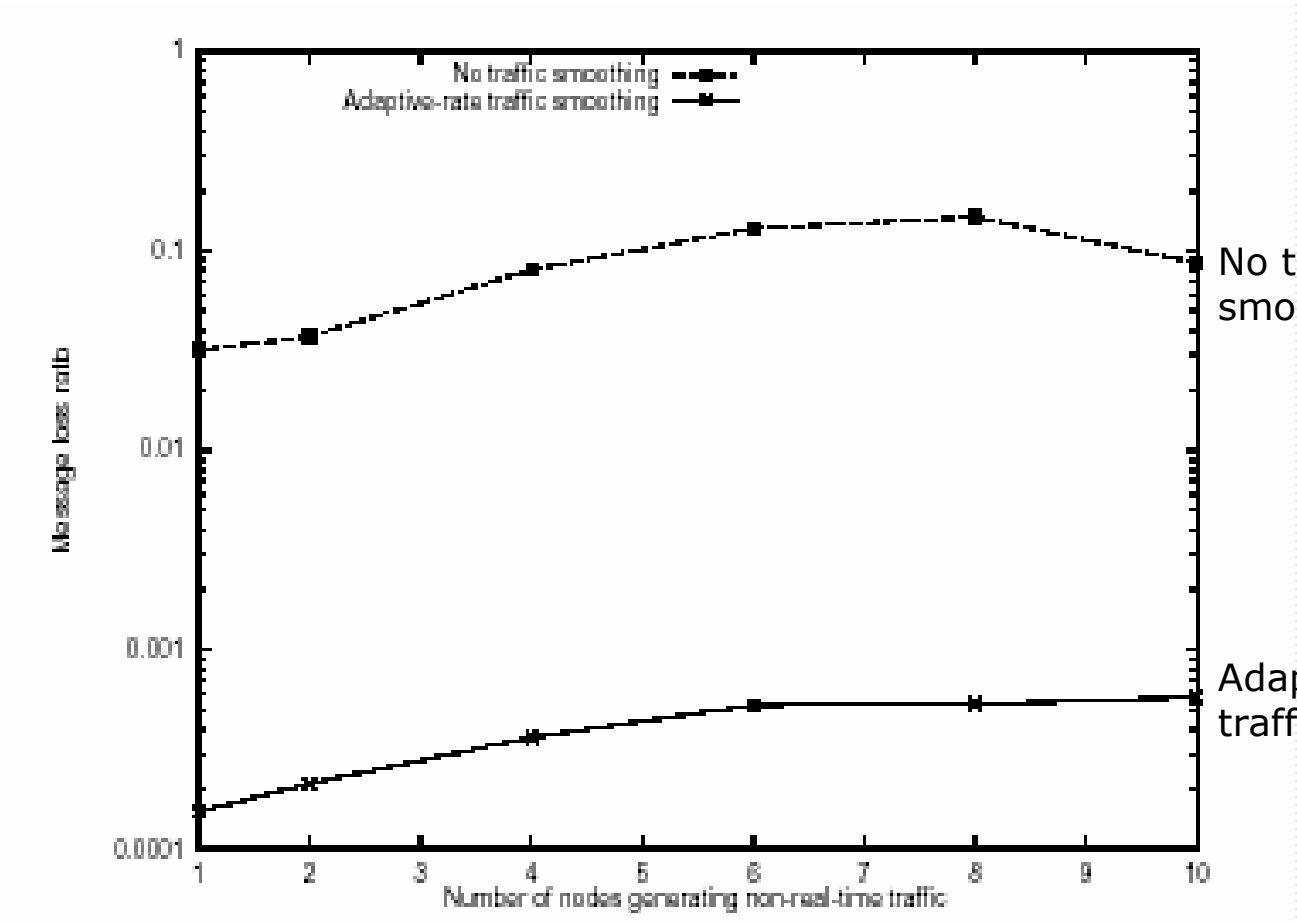


No traffic smoothing



Adaptive-rate traffic smoothing

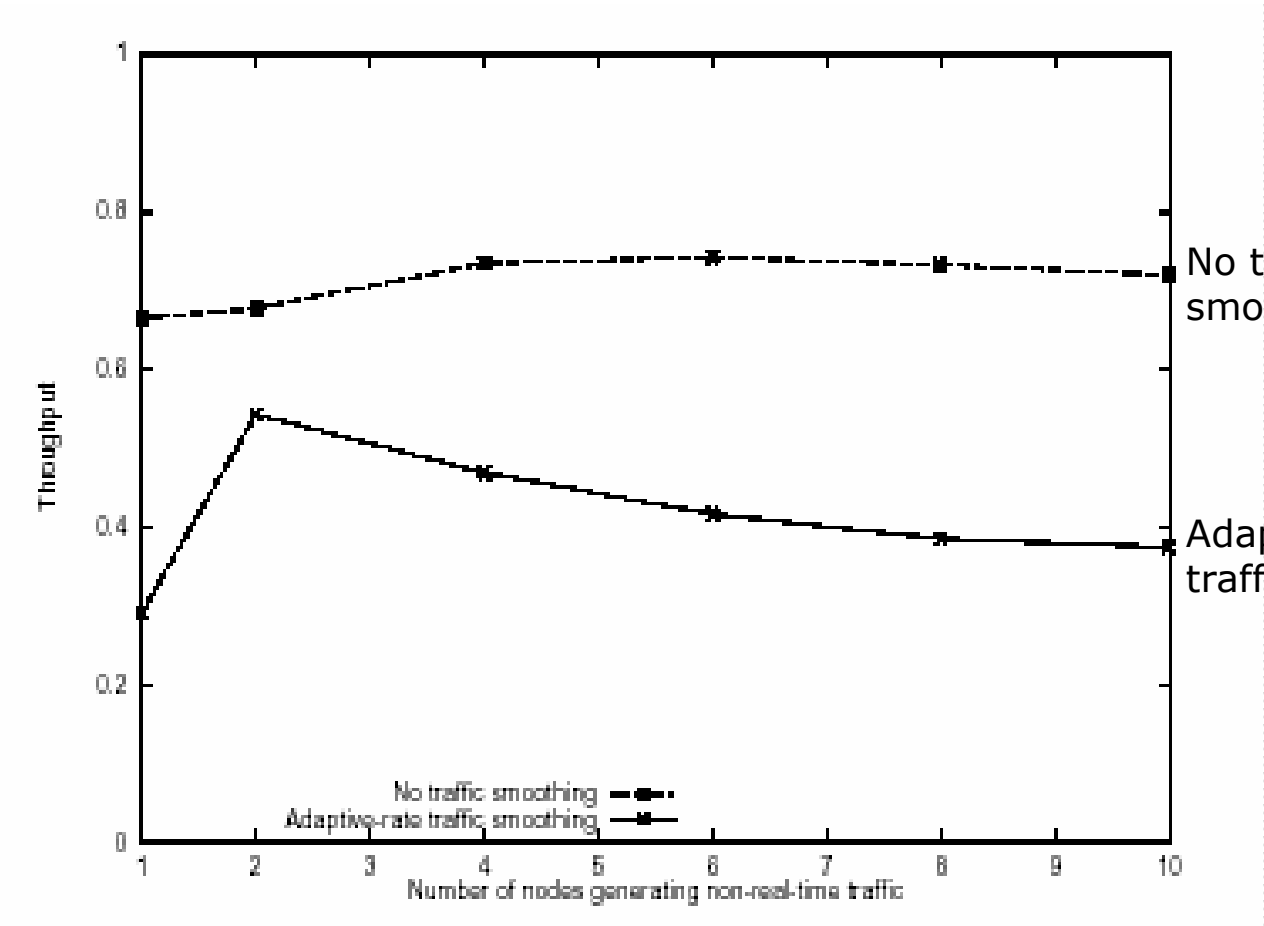
Experiment Result: loss ratio of RT traffic in greedy mode



No traffic smoothing

Adaptive-rate traffic smoothing

Experiment Result: Throughput for non-RT traffic in greedy mode



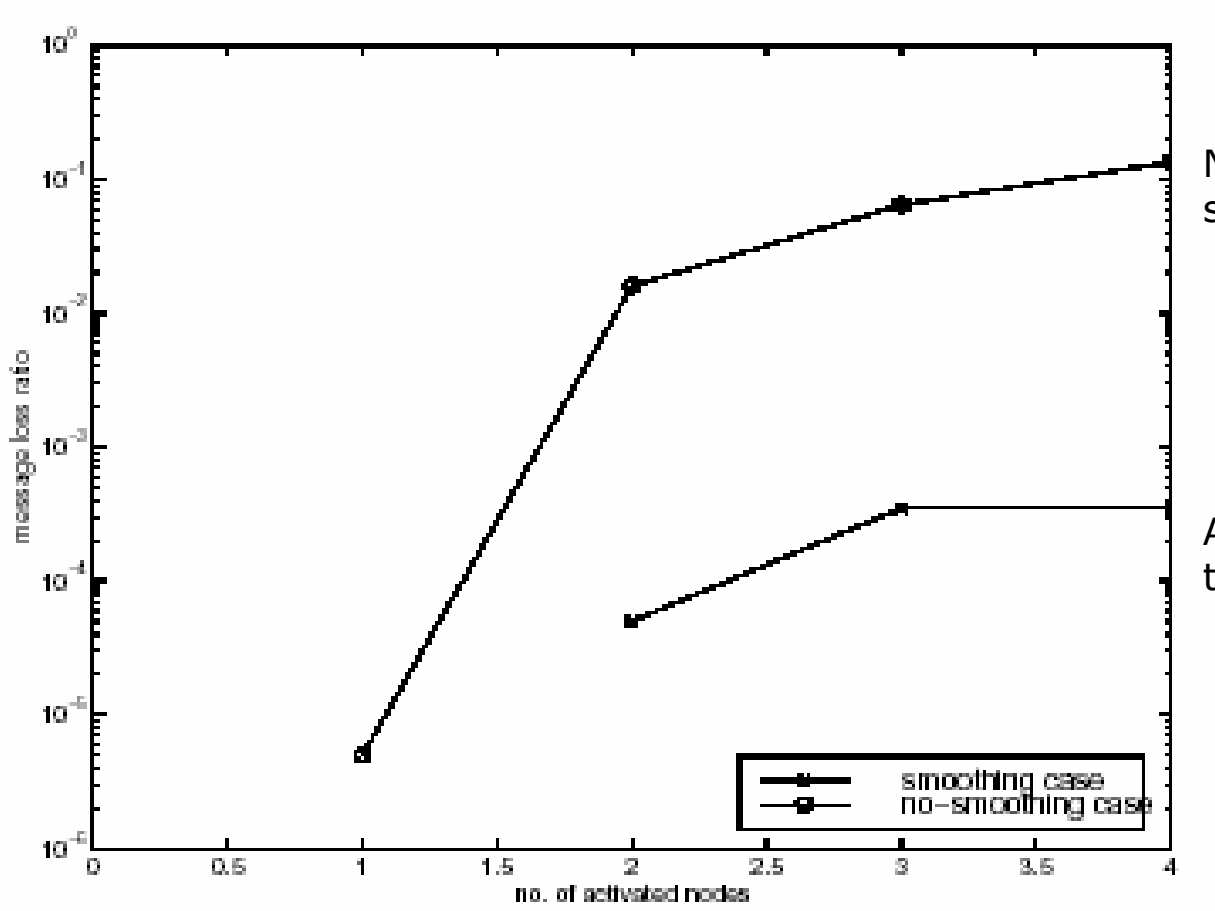
No traffic smoothing

Adaptive-rate traffic smoothing

Experiment Evaluation on Windows NT

- Windows NT is widely deployed
- Implemented using NDIS
- Acquire packet collision through NT system call
  - Frequent call can freeze the system
  - Result in less responsiveness compared with Linux version
  - Can change both CBD and RP

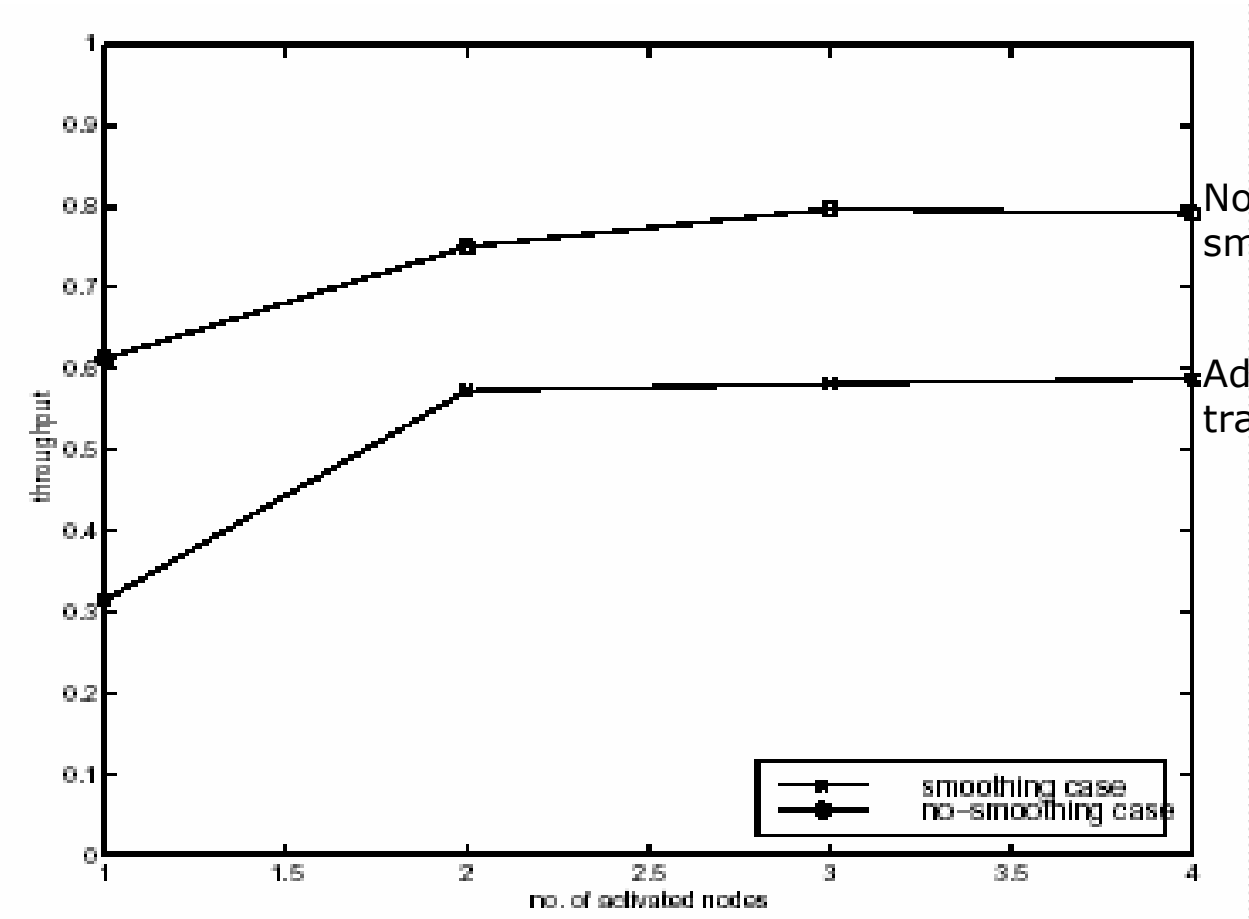
Experiment Result: RT message loss ratio (greedy mode)



No traffic smoothing

Adaptive-rate traffic smoothing

Experiment Result: non-RT traffic throughput



Conclusion

- Traffic Smoother provides soft real-time communication service
- Regulate bursty TCP/IP traffic to be smooth stream
- Adapt traffic generation rate to current network condition
- Provide good throughput to non-RT traffic while meeting RT traffic requirements
- Further extension for other applications such as real-time video