QCN: Transience, Equilibrium, Implementation

Abdul Kabbani, Ashvin Lakshmikantha, Rong Pan, Balaji Prabhakar, Mick Seaman

Outline of presentation

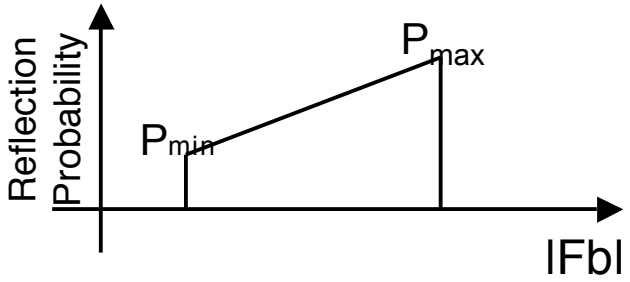
- Brief review of QCN
  - 2-pt and 3-pt versions
  - Performance and deployment
- Equilibrium and Scalability
- Transience
- Conclusions

Basic QCN

- 2-point architecture: Reaction Point -- Congestion Point

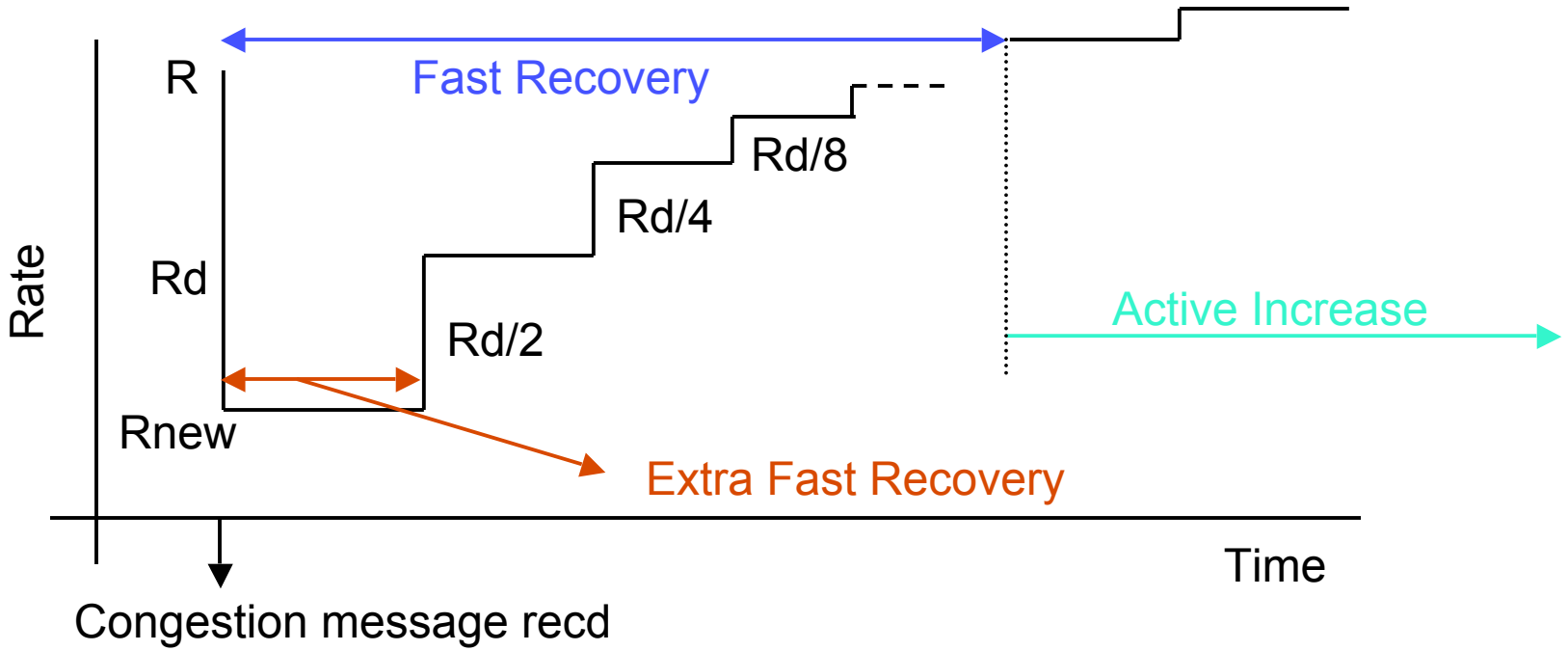
1. **Congestion Points:** Sample packets, compute feedback (Fb), quantize Fb to 6 bits, and reflect only *negative* Fb values back to Reaction Point with a probability proportional to Fb.

$$Fb = -(q_{off} + w\, q_{delta})$$
$$= -(\text{queue offset} + w.\text{rate offset})$$



2. **Reaction Points:** Transmit regular Ethernet frames. When congestion message arrives: perform multiplicative decrease, fast recovery and active increase.
   - Fast recovery similar to BIC-TCP: gives high performance in high bandwidth-delay product networks, while being very simple.

Fast Recovery and Active Increase



Basic QCN: Outcomes/results

- Easy to deploy, light resource requirement
  - No header modifications, no tags, **immediately deployable**.
  - Can work with a *single* rate limiter.
    - Alias all flows which have received negative feedback onto the rate limiter. RL becomes “meta-flow” with fast recovery + active increase ensuring good performance.
    - The algorithm is well-defined; i.e. does not rely on the existence of multiple rate limiters for correctness of specification since it has no tags or probes.

- Quantizing Fb simplifies implementation
  - Fb value used to index into a small table to find the decrease factor.
    - No potentially expensive hardware resources needed for computations.
  - Lookup table also makes the scheme **easily reconfigurable** (if Fb --> Rate relation changes), a useful workaround.

QCN: 3-point architecture

- ReaP--CP--RefP
  - Allows signaling Fb=0 values to ReaP, which indicate *lack* of congestion. Only the RefP can do this without the use of RP-->CP association tags.
  - When a ReaP receives an Fb=0 signal, it just skips to the next cycle of Fast Recovery or Active Probing; i.e. it increases the rate appropriately and it restarts the byte counter
    - Simple behavior, no increase gains or parameters.

- Single bit needed for signaling Fb=0, call this the Fb0 bit
  - We can use the DE (Discard Eligible) bit as the Fb0 bit

Drift

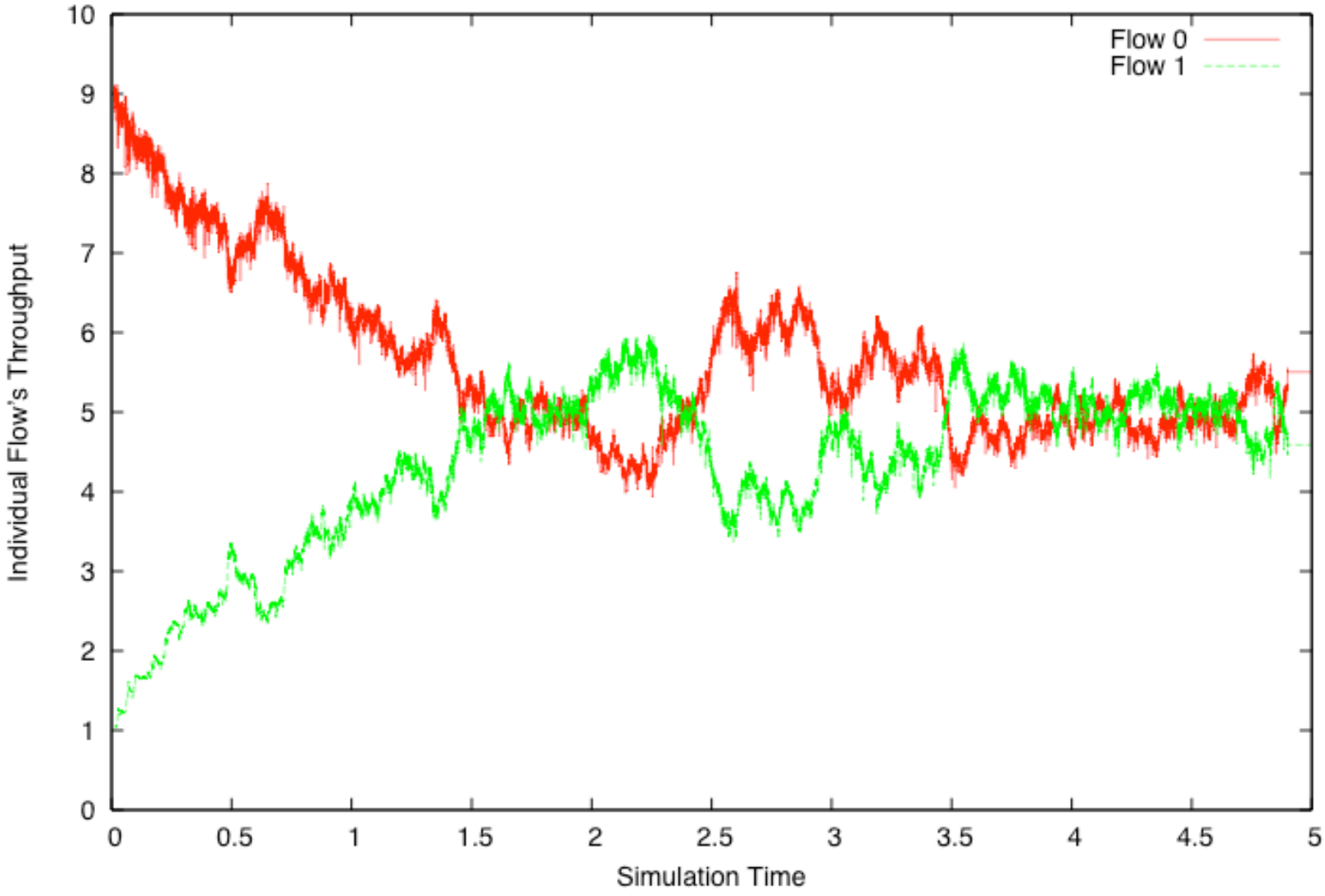
- Since both Fast Recovery and Active Increase use ***byte counters*** for self-clocking, it is advisable to have a ***time-driven*** "rate drift"
  - Provides failsafe operation, allows rate limiters to be decommissioned

- Drift
  - Drift clock corresponding to RL expires every T units of time
  - When clock expires
    - Increase transmission rate from R to R.X, where X > 1
    - Restart clock
  - Any time an Fb<0 signal is received by RL, restart the drift clock
    - Note: this ensures drift is used only minimally and when network is uncongested
    - Also note it makes drift inversely proportional to a flow's sending rate, since larger sources get more Fb<0 signals relative to small sources

- Notes on Drift
  - It brings about quicker convergence to fairness
  - Because it is time-based, not packet-based, it can also be very helpful in grabbing extra available bandwidth (more on this later)

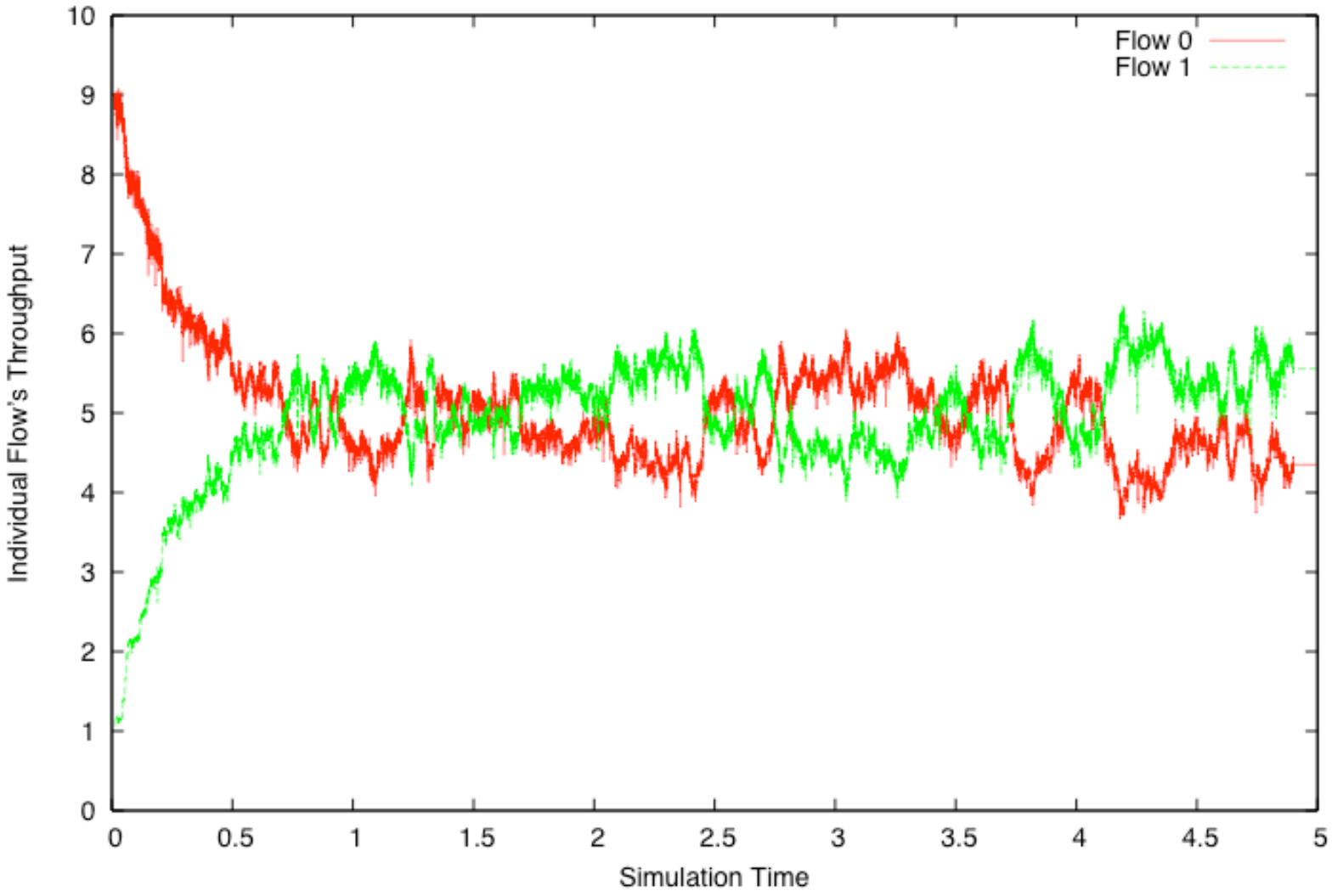
Fairness: No Drift

2 flows: 1 starting at 1Gbps, 1 starting at 9 Gbps



Fairness: With Drift

2 flows: 1 starting at 1Gbps, 1 starting at 9 Gbps

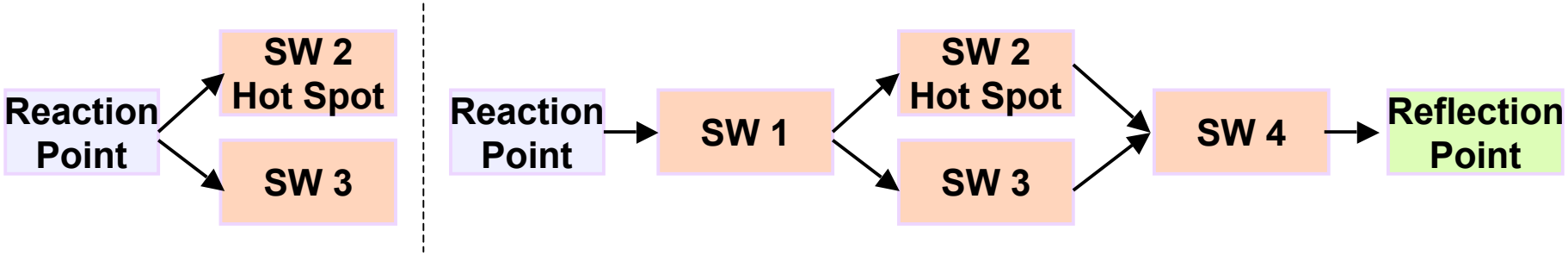


2-pt vs 3-pt QCN

- 3-pt QCN performs better than 2-pt QCN
  - All simulations indicate this
    - Main reason is that the input rate can be matched to the output rate quickly
  - However, in most normal cases of operation, the improvement is marginal
    - Improvement is significant when grabbing a lot of excess bandwidth

- But the 3-pt QCN has a problem when flows share RLs, which occurs when the number of RLs is small or there is multipathing
  - Basically, signaling rate increases requires path information
  - As already shown in an ad hoc meeting, 2-pt QCN has no problem when RLs are shared

Discussion of 3-point architecture: Signaling rate increases



- **Problem:** Imagine SW 2 is congested, but SW 3 has bandwidth to spare. Probing or forward signaling will bring fluctuating positive and negative signals.
  - Cannot obey both signals because (a) hot spot will be overloaded, (b) positive signals will be *more numerous.*
- Disambiguation of the signals requires path knowledge at either the ReaP or the RefP.
- If we used something like a CPID or other path info to get around this
  - There is a potential "stuck at low rate problem." That is, it is quite likely that the CPID at the ReaP will be that of SW 2. If the flow passing through SW 2 terminates, then the ReaP has stale CPID. Specifically, this causes the ReaP to ignore any positive signals from SW 3 and it has to rely on Active Increase to bring its rate up, rendering positive signaling ineffective.
- Conclusion
  - 2-pt QCN works reasonably well in all cases
  - We only need to improve its performance in terms of grabbing extra bandwidth; and we have a simple idea for doing this; more on this point later

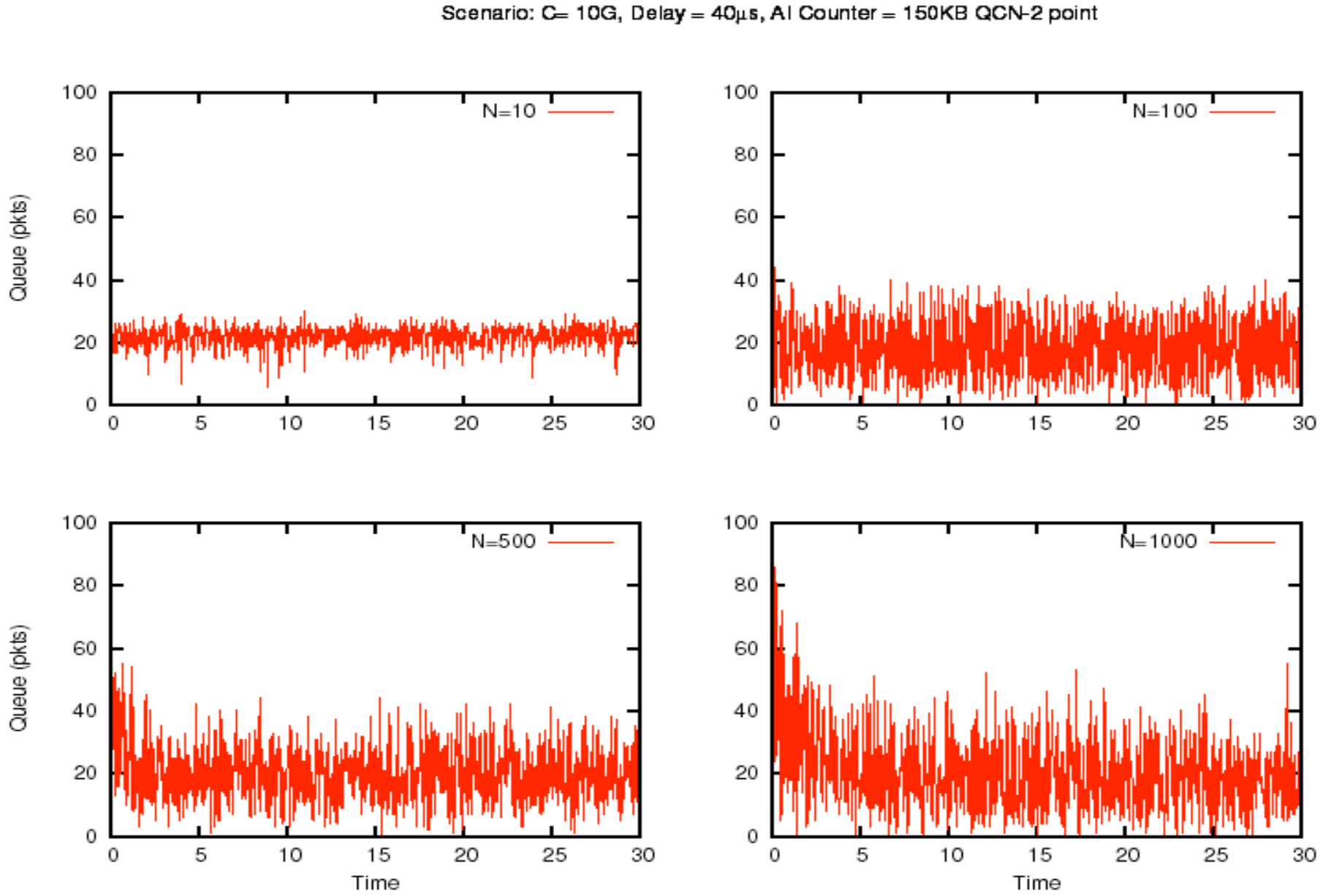
Part 2: Scalability, Stability

Scalability Analysis

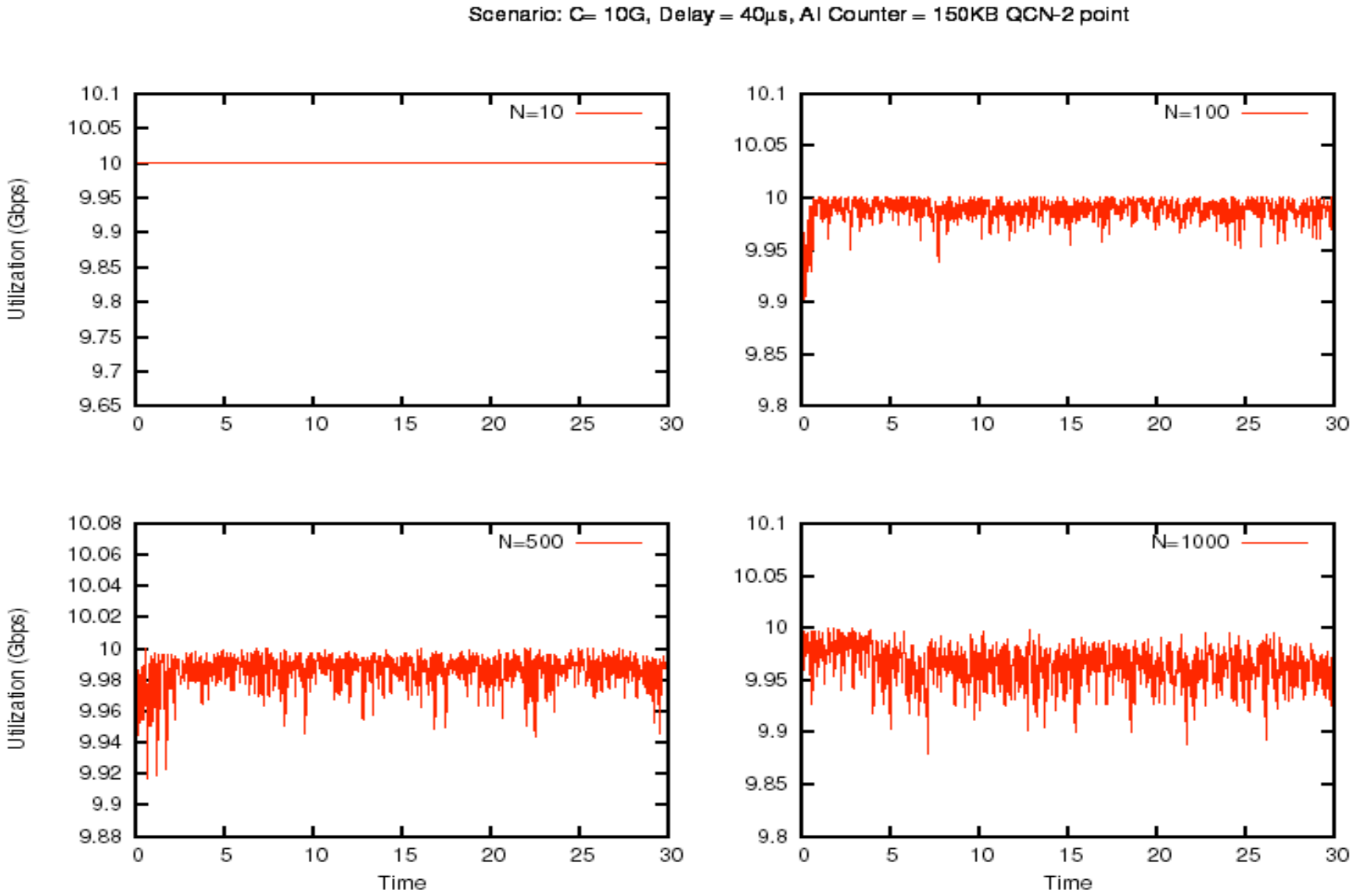
- This refers to understanding the behavior of QCN when N, the number of sources, gets very large; some questions
  - Do we need to choose parameters carefully?
  - Is there a problem with aggressing modes of active increase, where new work is injected into the system?
  - Towards a theoretical model…

- We simulate 2-pt QCN with N sources, where N varies from 10 to 1000
  - We look at the queue behavior and the link utilization
  - We use the following fixed set of parameters for *all* values of N
  - Queue size: 100 (1500 B) packets; Qeq = 22; Gd = 128; w = 2; A = 12 Mbps; Drift: X = 1.005, T = 500 microsecs; Sampling function = linearly increases with |Fb| from 1--10%; RTT = 40 microsecs
  - The *aggregate* starting rate of sources = 100 Gbps
    - I.e. a source starts at 100/N Gbps
  - We use AI byte-counter values of 100 pkts and 25 pkts

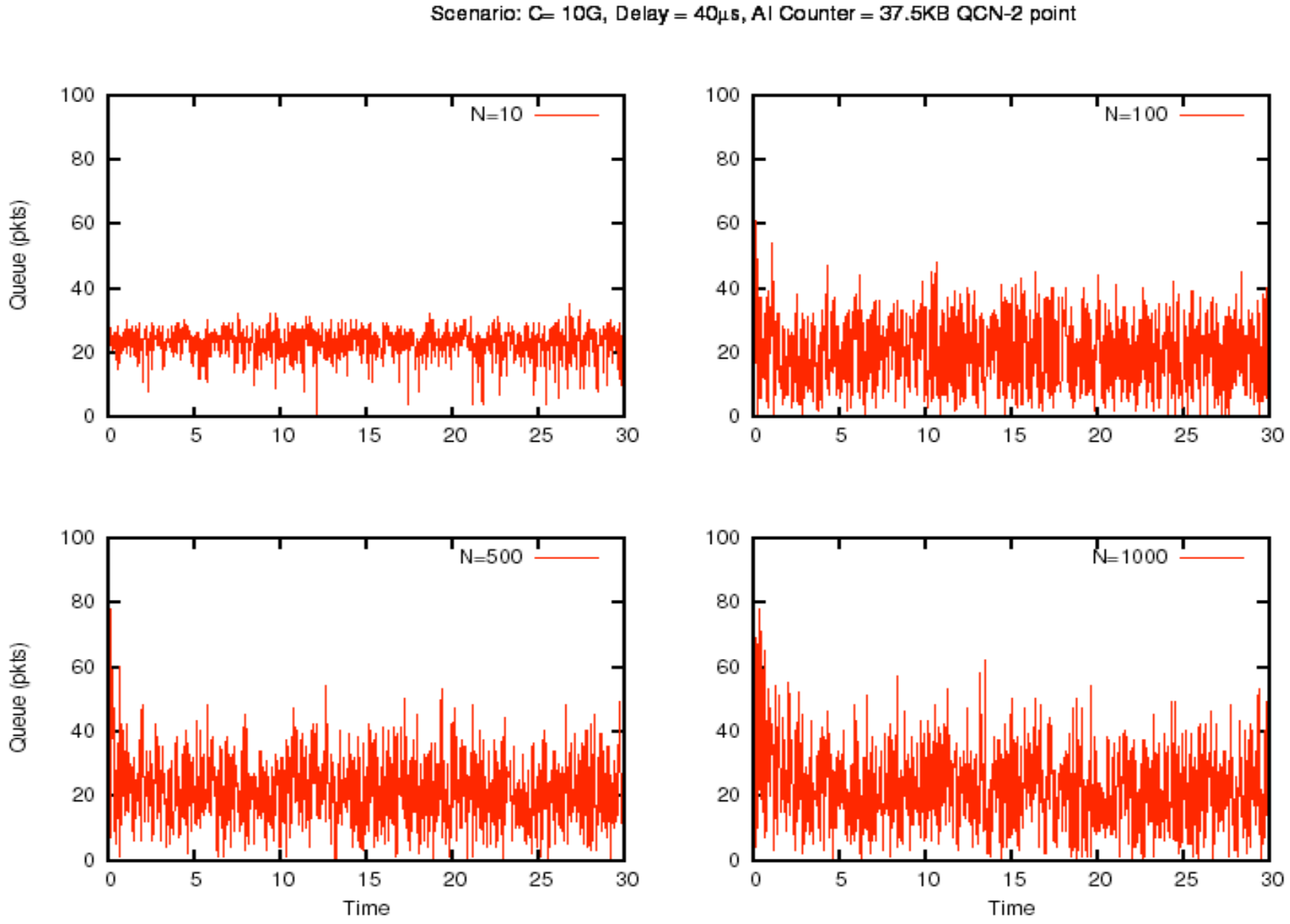
Scalability Analysis: Queue Size
AI byte counter = 100pkts



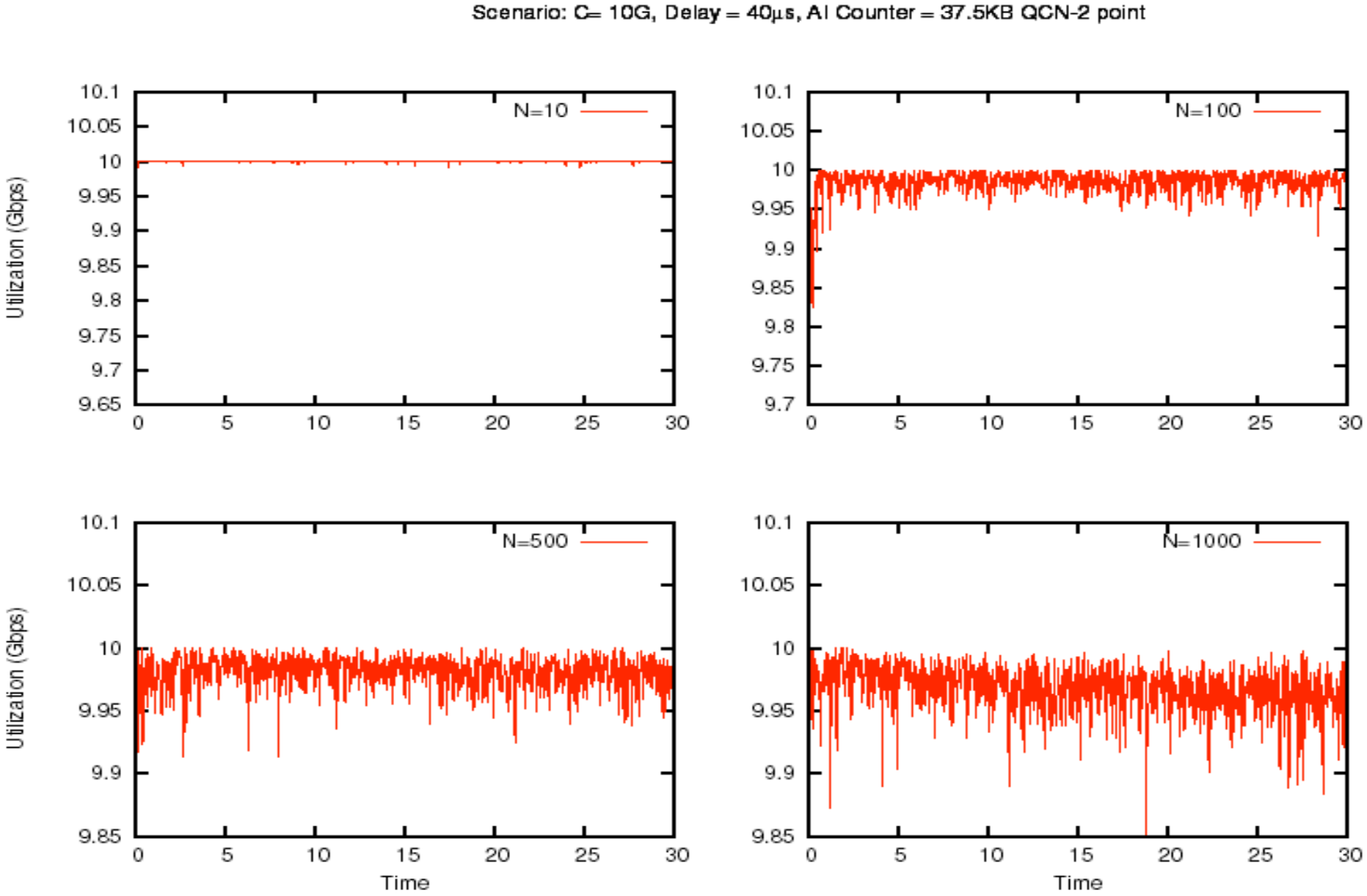
Scalability Analysis: Rate AI byte counter = 100pkts



Scalability Analysis: Queue size
AI byte counter = 25pkts



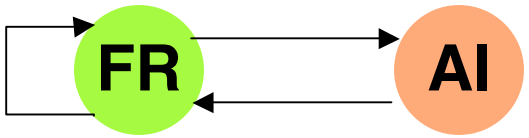
Scalability Analysis: Rate AI byte counter = 25pkts



Scalability Analysis: Inferences

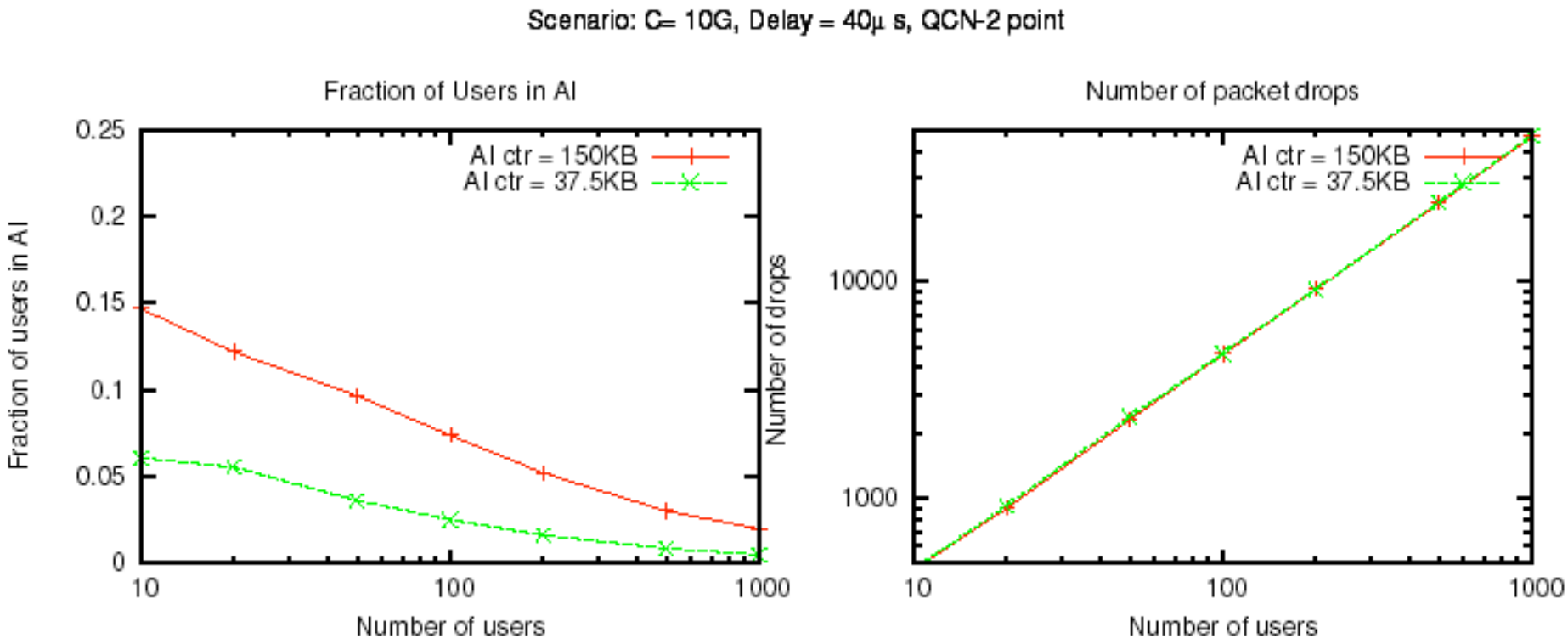
- 2-pt QCN has excellent scalability properties
  - Using exactly the same parameters across a large range of N causes no real scalability issues; why?

- Each source is in FR or AI at any time, as shown in the state diagram



- When there are N sources, the fair share rate per source equals C/N
  - The operating rate goes down with N
  - Therefore, the time spent in FR (which is based on using byte-counters) *increases* with N
  - The *more* sources there are, the *longer* each sources spends in FR
  - Hence, the amount of new work injected into the system during AI, the *potential cause for instability*, remains bounded; in fact, it decreases with N
  - This gives 2-pt QCN its stability property!

Scalability Analysis



- Conclusion
  - Aggressive forms of AI don't hurt stability as N increases
  - This observation will be very useful for grabbing extra bandwidth

Part 3: Transience

Transient Behavior

- Since the steady-state behavior of QCN is good as N increases, it is worth looking into transient behavior; we consider two types of transient situations
  1. Serious bottleneck appears, or there is a heavy oversubscription
     - Need to ensure that the sources quickly find the new *lower* rate
  2. Serious bottleneck disappears, or this is a lot of extra bandwidth
     - Need to ensure sources quickly grab the extra bandwidth
     - Especially for 2-pt QCN (3-pt QCN is ok here, but cannot really be used)

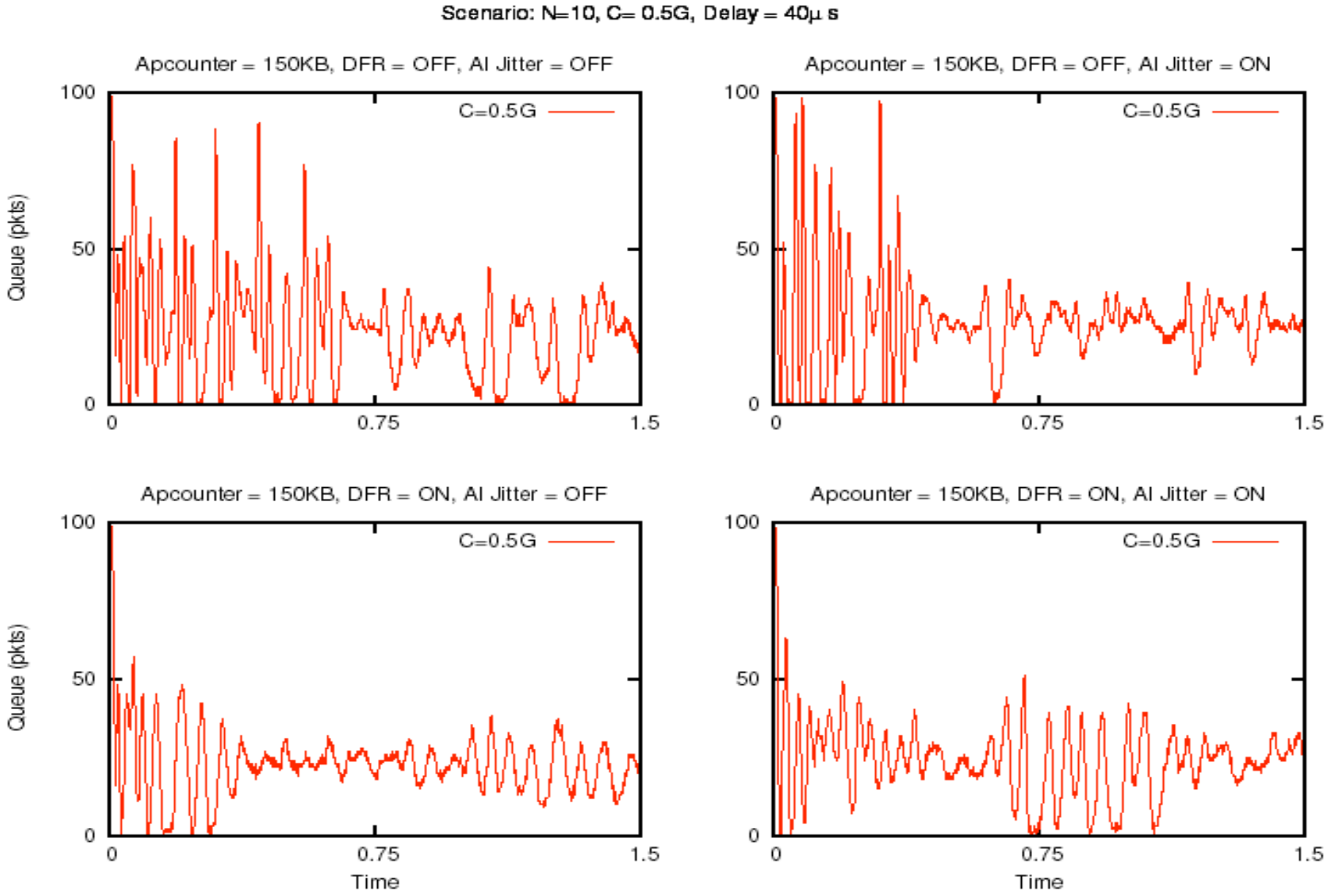
- We propose some tweaks to 2-pt QCN to cope with Case 1; we discuss a simple tweak for Case 2
  - These tweaks take simple advantage of extra information available in negative feedback
  - Our goal is to retain the simplicity of 2-pt QCN while improving its performance

Case 1: Severe Bottleneck

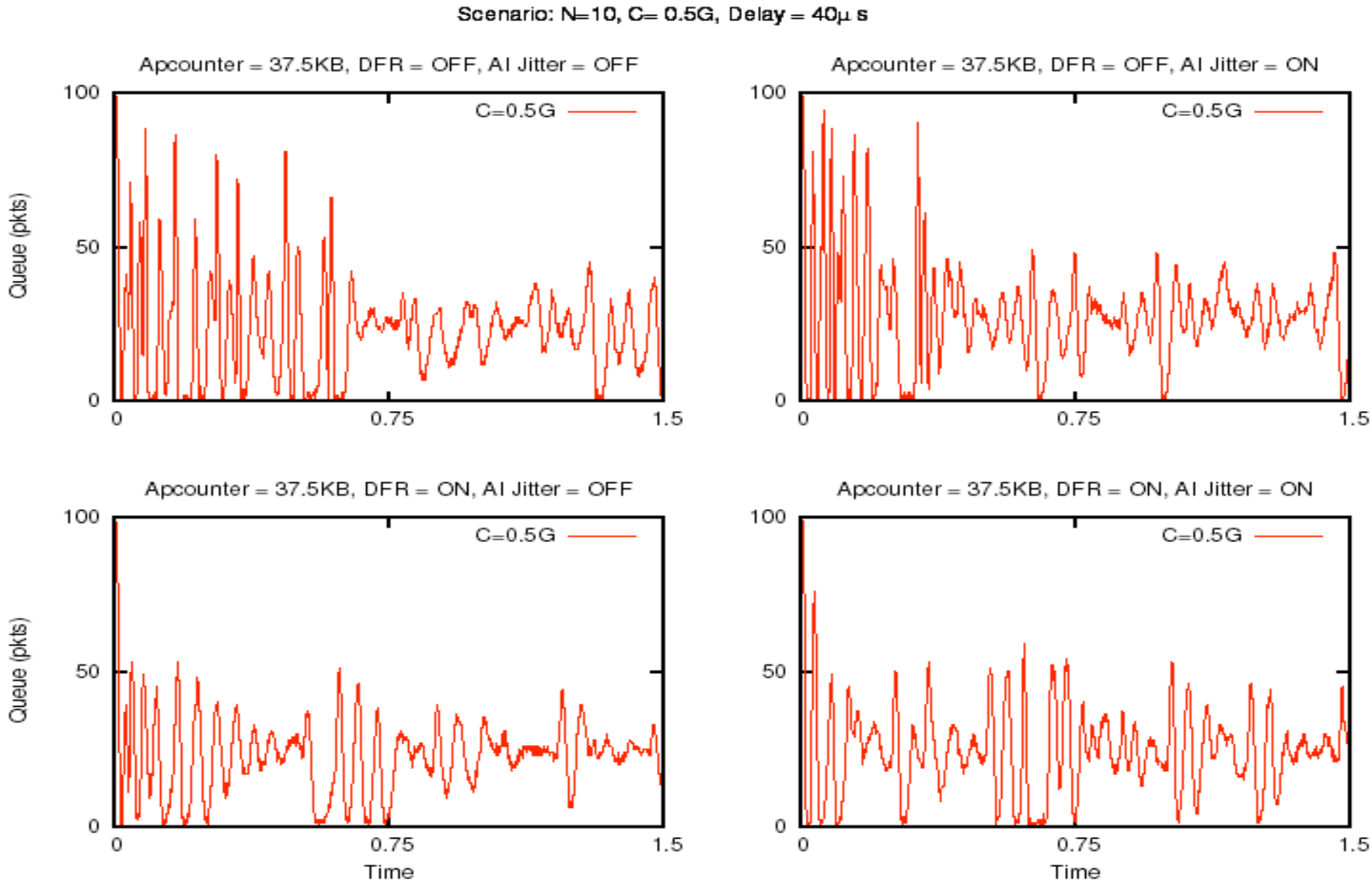
- Discounted Fast Recovery: To help sources achieve the lower rate quickly
  - Observation: This is detectable at the source because it leads to many bursty negative Fb signals at ReaP.
  - Idea: Discount the recovery.
  - Algorithm: If number of negative Fb signals in first cycle of FR less than or equal to 5, recover Rd/2, else recover Rd/4.

- Jittering FR and AI byte-counters: Minimizes synchronization effects when sources react to bottleneck simultaneously
  - Algorithm: Let FR byte counter increment at a random number chosen in the range, say, of [105 KB, 195KB]. The mean is 150KB or 100 pkts.
  - Do the same for the AI byte-counters.

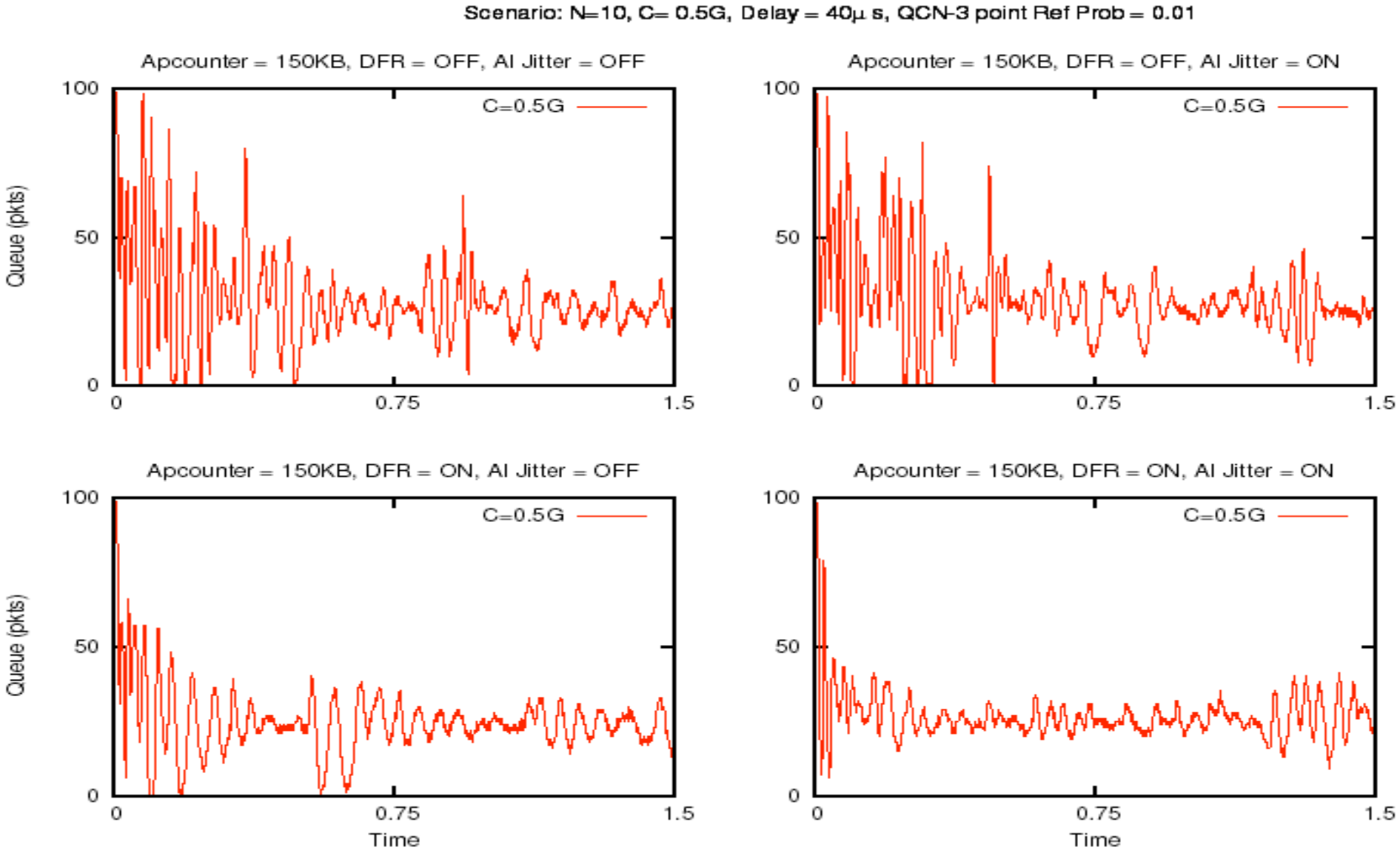
2-pt QCN: 0.5G, AI counter = 150KB



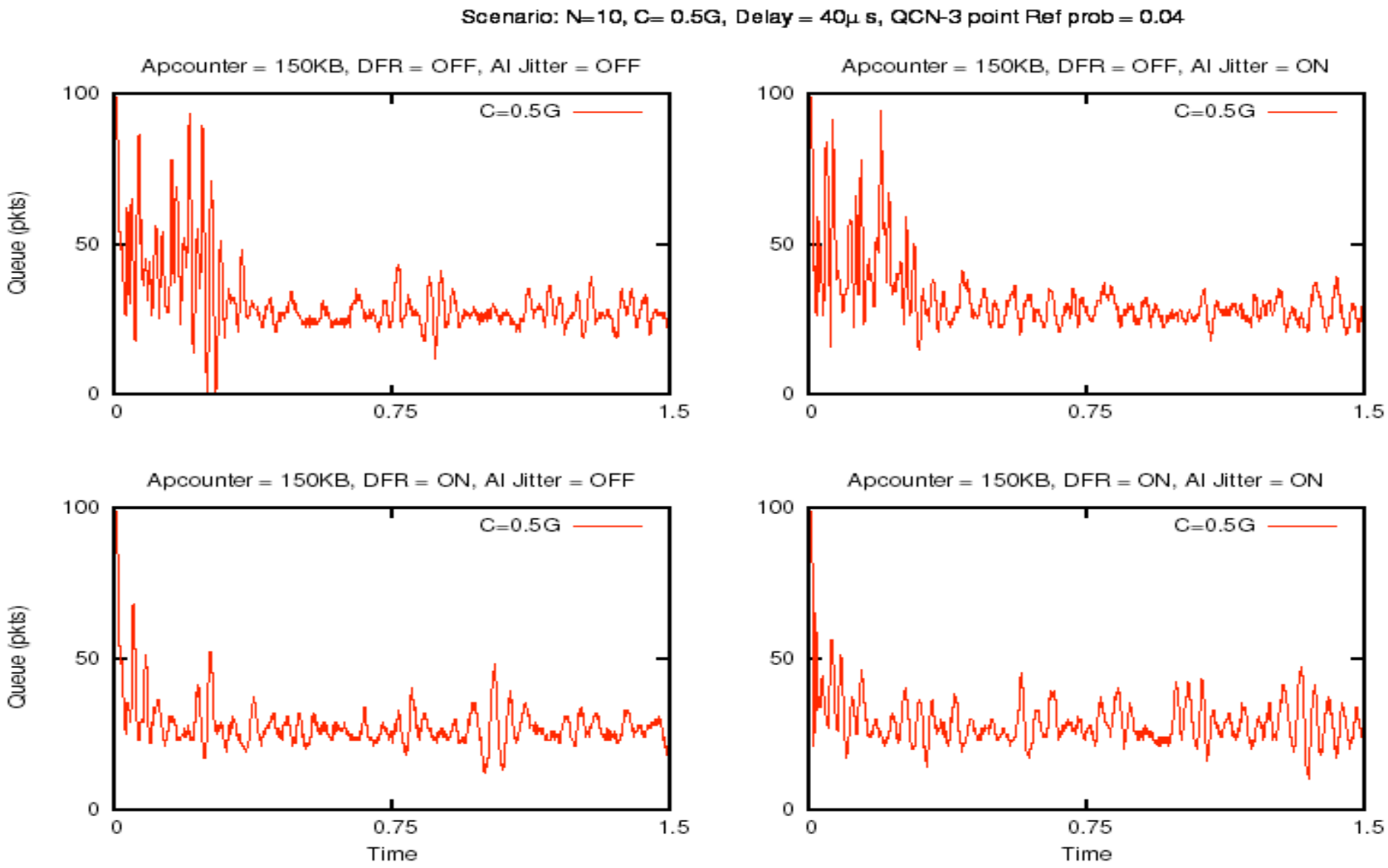
2-pt QCN: 0.5G, AI counter = 37.5KB



3-pt QCN: 0.5G, AI counter = 150KB Reflection Prob = 1%



3-pt QCN: 0.5G, AI counter = 150KB Reflection Prob = 4%



Case 2: Grabbing Extra Bandwidth

- The way we have defined AI in QCN, we have used byte counters to increase rate
  - The rule: Every 100 (or 25) pkts R = R + 12 Mbps
  - This gives rise to an exponential increase in rate over time:

$$R(t + k\ pkts) - R(t) = 12Mbps$$

$$\Longrightarrow \frac{R(t + k\ pkts) - R(t)}{Tx\ time\ for\ k\ pkts} = \frac{12\ Mbps}{Tx\ time\ for\ k\ pkts} = \alpha R(t)$$

$$Or,\ \frac{dR}{dt} = \alpha R(t) \quad \Longrightarrow \quad R(t) = R_0 e^{\alpha t}$$

  - So, the rate increases *exponentially* during AI
  - However, as pointed out in several simulations (thanks Cyriel, Davide and Mitch), AI is not be quick enough in grabbing extra bandwidth when a severe bottleneck disappears; that is, it is worse than exponential

- Why?

Case 2: Grabbing Extra Bandwidth

- Consider 10 sources bottlenecked to 0.5G as in the single hop output-generated hotspot (see Davide's presentation for details)
  - When bottlenecked, the rate of each source is small; e.g. 50 Mbps
  - Moreover, the offered load per source may not be high
  - Therefore, the AI byte-counters which advance the rate actually have no bytes to count; this kills exponential growth
  - Need packets to increase rate!

- A possible simple solution (already proposed in the Orlando meeting)
  - Use timer-based increase during AI; avoid byte-counters here, although we should still use them for FR (because that is precisely why we got scalability)
  - Timer-based multiplicative increase, as in drift, gives exponential rate increases over time *regardless* of whether we have packets or not
  - Need to work out the details, but the basic problem of not having packets to use for AI is removed by the use of timers

Conclusions

- We have proposed and studied 2-pt and 3-pt QCN
  - 3-pt QCN performs slightly better than 2-pt QCN in most common cases of operation, but has implementation issues when flows share RLs
  - So 2-pt QCN (which is a combination of BIC and REM) seems the preferred algorithm of the two
- We have seen that 2-pt QCN has excellent stability properties when the number of sources increases
- It's transient behavior can be improved with the help of minor tweaks to the basic scheme; we shall present further work on this shortly
  - Many thanks to all who gave us feedback, esp to Davide, Cyriel, Mitch for pointing out the single hotspot output generated scenario and other discussions