Max Progressive Network Update

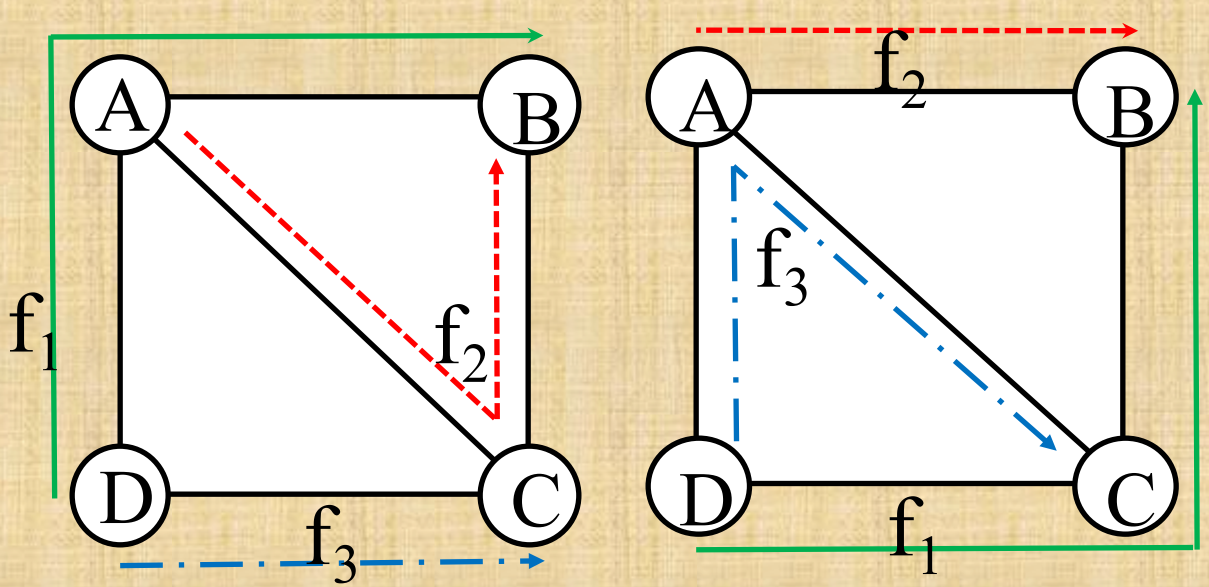
Yang Chen and **Jie Wu**

Temple University, USA

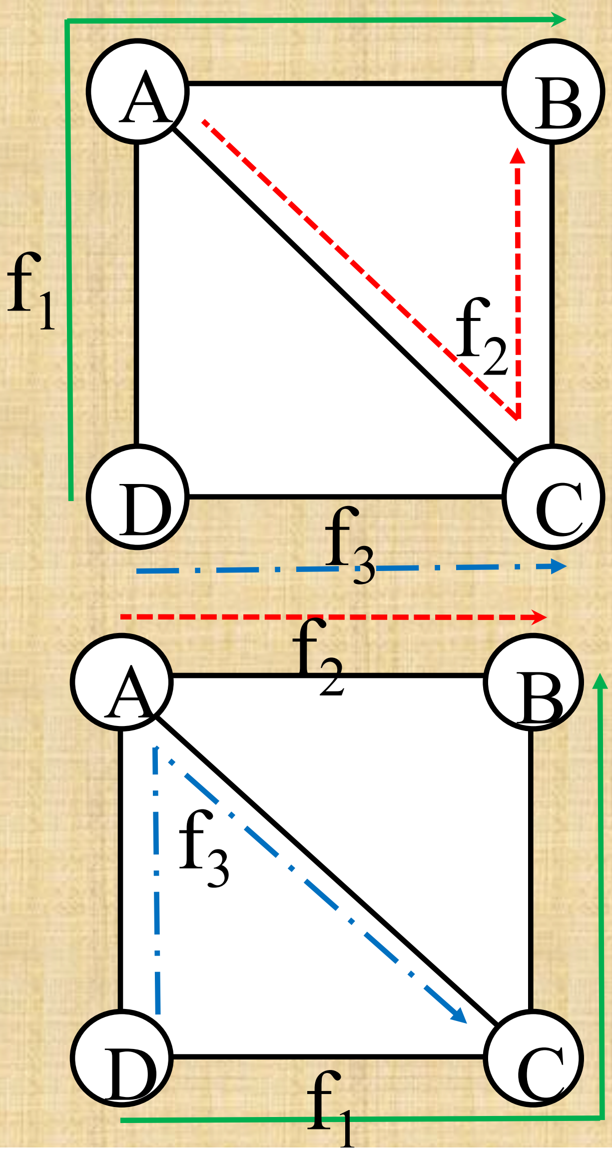
SDN Network Update

- **Network update**
  - Can adapt to frequent traffic changes for high network utilization.
- **Challenges**
  - Rule updates from the controller to the individual switches traverse an asynchronous network and may arrive out-of-order.
- **Objectives**
  - Optimality, consistency, and swiftness
- **Basic update methods**
  - Ordering update protocols
  - Two-phase update protocols
- **In our paper**
  - We use switch buffer to assist the update in order to migrate flows consistently.

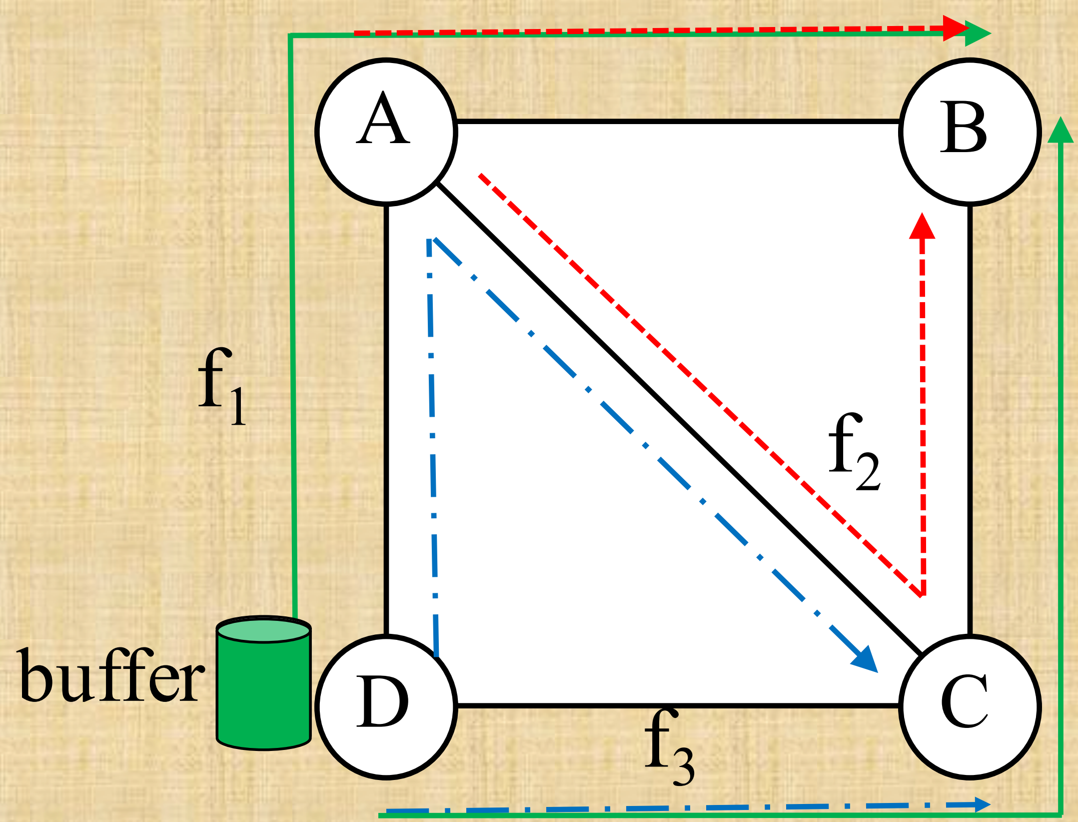
Motivation



- **Consistency**
  - Loop-free
  - Drop-free
  - Congestion-free



Switch buffer (for swiftness)



Problem Formulation

- **Problem**
  - Given the initial and final network states, we need to find a feasible solution to consistently migrate flows.

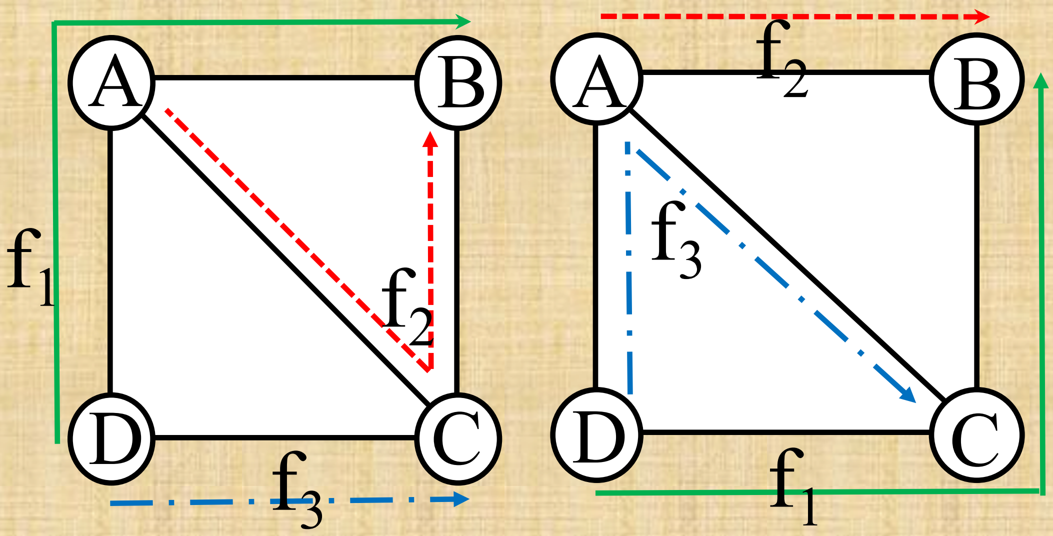
(Finding the optimal update schedule is NP-hard with the constraint of link capacity.)[1]

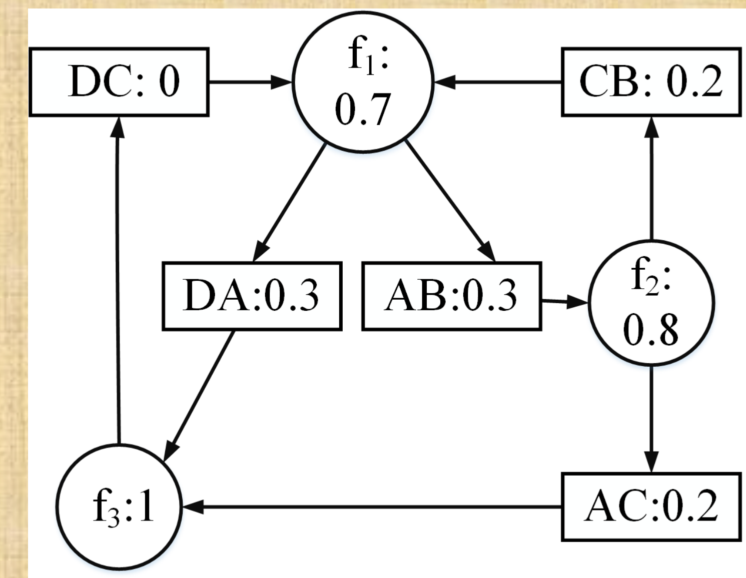
- **Objective**
  - Find the quickest update schedule with the help of switch buffer: balance between updating time and buffer size

- **Definitions:**
  1. Dependency graph
  2. In degree and out degree of a flow node
  3. Necessary condition for deadlocks: cycles among flows and link resources

[1]: "Dynamic Scheduling of Network Updates", SIGCOMM14

Max Progressive Updating Method (MAPUM)



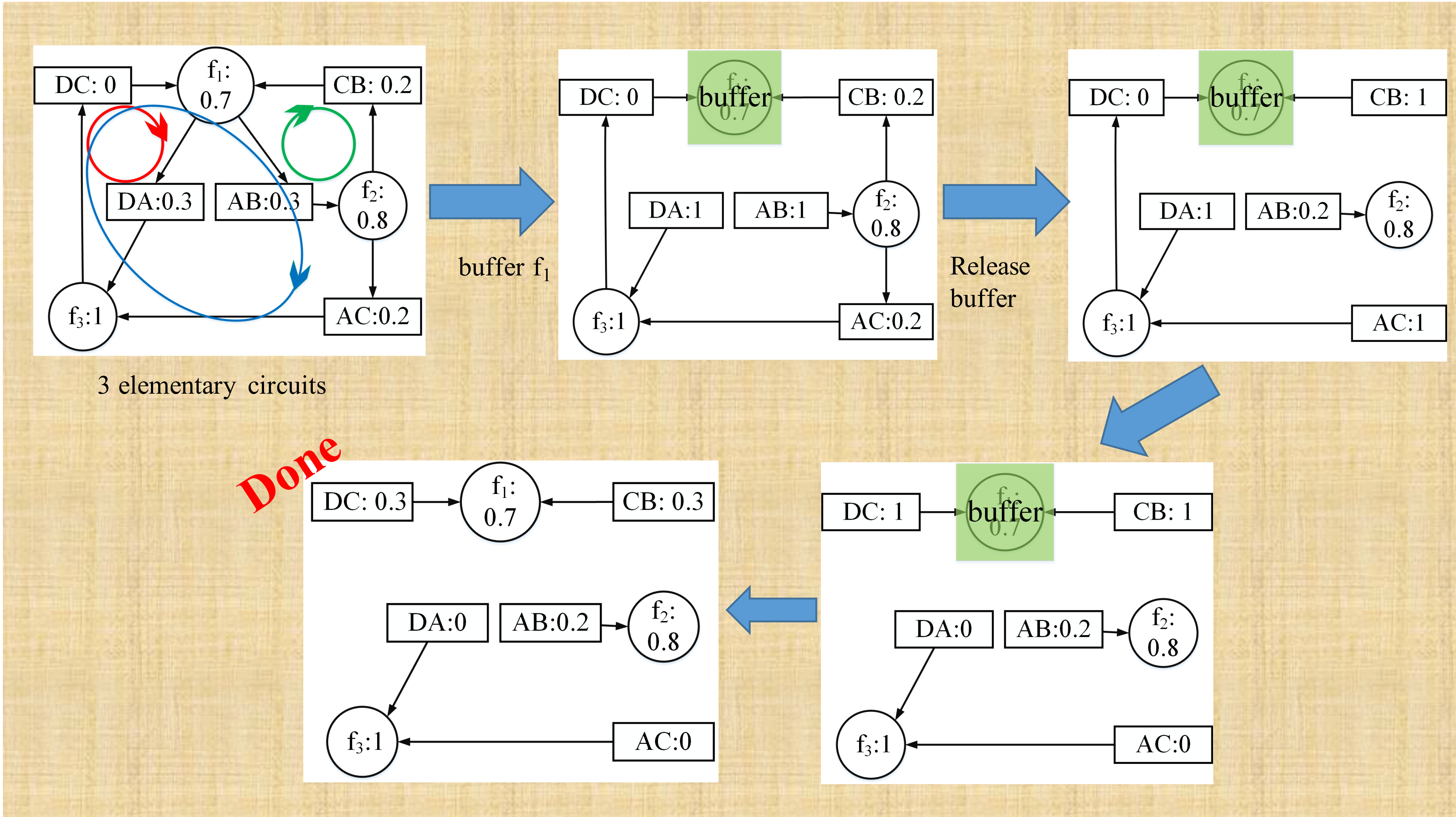


Link's capacity: 1; Flow demand: f_1=0.7; f_2 =0.8; f_3=1

Dependency graph

- If the dependency graph is a DAG, then there are no deadlocks; otherwise, limit flows to break all elementary cycles.

D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM 2006 Journal on Computing*.


DC: 0
f1: 0.7
CB: 0.2
DA:0.3
AB:0.3
f2: 0.8
f3:1
AC:0.2
3 elementary circuits
buffer f1
DC: 0
buffer
CB: 0.2
DA:1
AB:1
f2: 0.8
f3:1
AC:0.2
Release buffer
DC: 0
buffer
CB: 1
DA:1
AB:0.2
f2: 0.8
f3:1
AC:1
DC: 1
buffer
CB: 1
DA:0
AB:0.2
f2: 0.8
f3:1
AC:0
Done
DC: 0.3
f1: 0.7
CB: 0.3
DA:0
AB:0.2
f2: 0.8
f3:1
AC:0

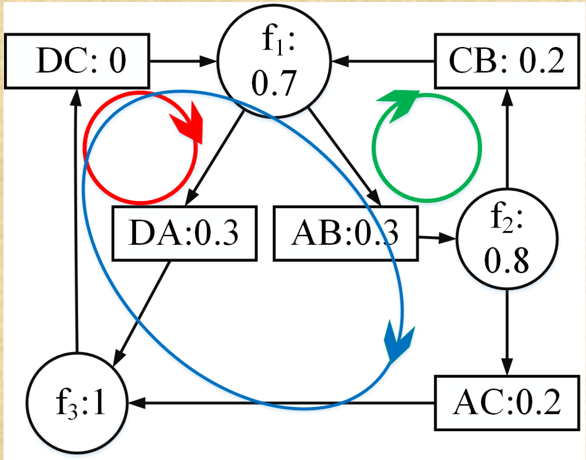
Max Progressive Updating Method (MAPUM)

- First use Dionysus (SIGCOMM14) to update flows until no more flows can be migrated any more.
- Remove potential deadlocks through rate-limiting flows

$$priority = \frac{degree(out)}{degree(in)} * \max delay(cycle_k)$$

(EMAPUM: $priority = \frac{degree(out)}{degree(in)} * \max delay(cycle_k) * b$)

- Select the highest priority flows to be buffered until all elementary cycles are resolved. (b is flow demand.)
- Release the buffer and migrate the buffered flows to the final states.



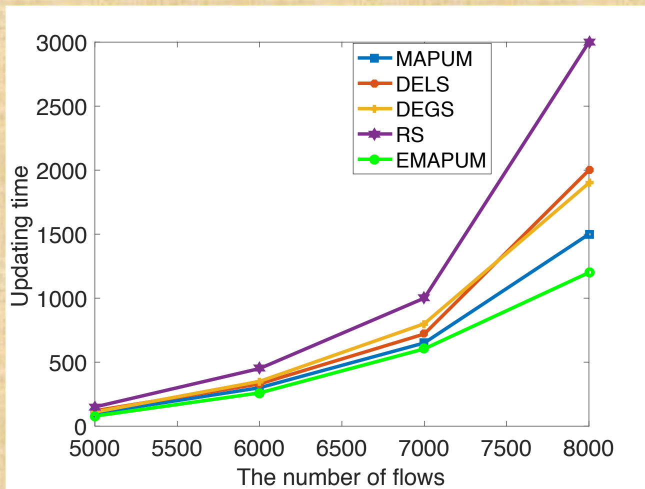
Deadlocks in the dependency graph (shown as three colored cycles)

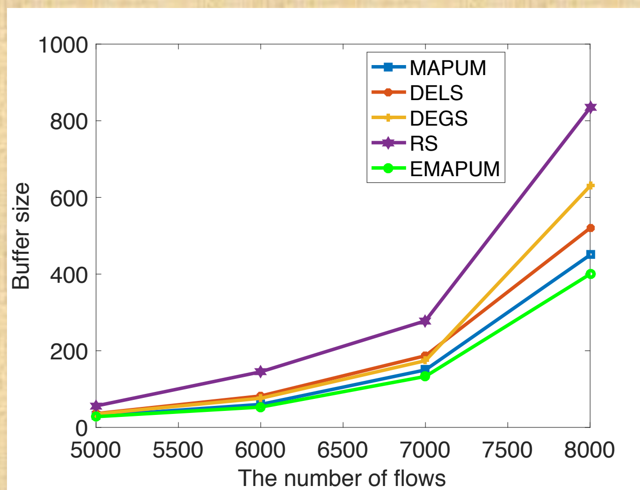
Evaluation

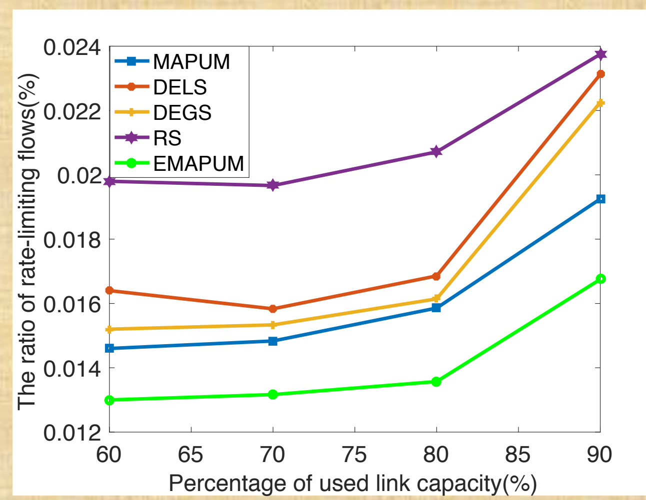
- We compare our MAPUM and EMAPUM with three schemes
  1. RS (random selection);
  2. DELS (delay-consideration);
  3. DEGS (degree-consideration).
- Measurement (assume one hop takes one time step)
  1. Updating time: from the first migration until all flows are migrated
  2. Buffer size: $\sum_{f \in F} t_f * b_f$

     (F: buffered flow set; t_f: time of f to be buffered; b_f: bandwidth of f)
  3. The number of rate-limiting flows

Evaluation Results







- Compared with RS, MAPUM and EMAPUM can reduce the updating time by 41% and 53%, respectively.
- In terms of buffer size usage, MAPUM and EMAPUM save over 37% and 42% buffer compared to RS.
- For ratio between rate-limiting and total flows, MAPUM and EMPUM are only 72% and 67% compared to RS.

Q & A