Learning-based Congestion Control Simulator for Mobile Internet Education

MobiArch 2021, New Orleans, United States

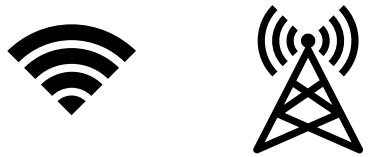
Junqin Huang, Linghe Kong, Jiejian Wu, Yutong Liu, Yuchen Li, Zhe Wang

Shanghai Jiao Tong University

Network congestion in mobile Internet

Heavy traffic burden from mobile devices:
Game, video, meeting, social network, etc.

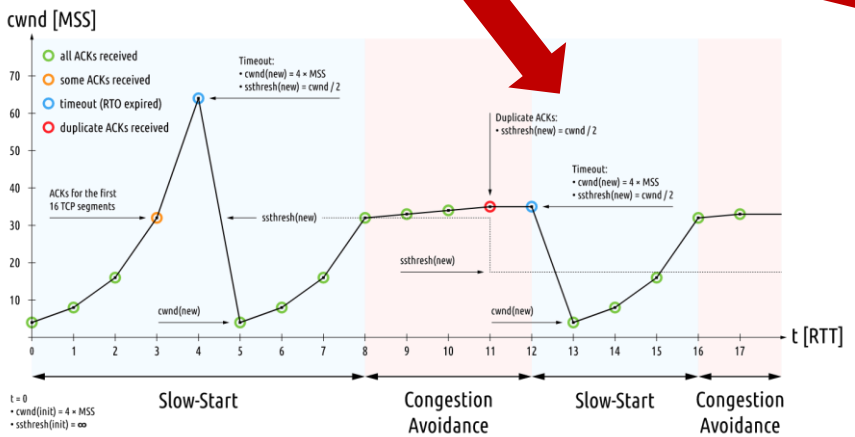
Highly dynamic network environment:
5G, Wi-Fi, statellite, optical fiber, etc.



Network congestion in mobile Internet

Heavy traffic burden from mobile devices:
Game, video, meeting, social network, etc.

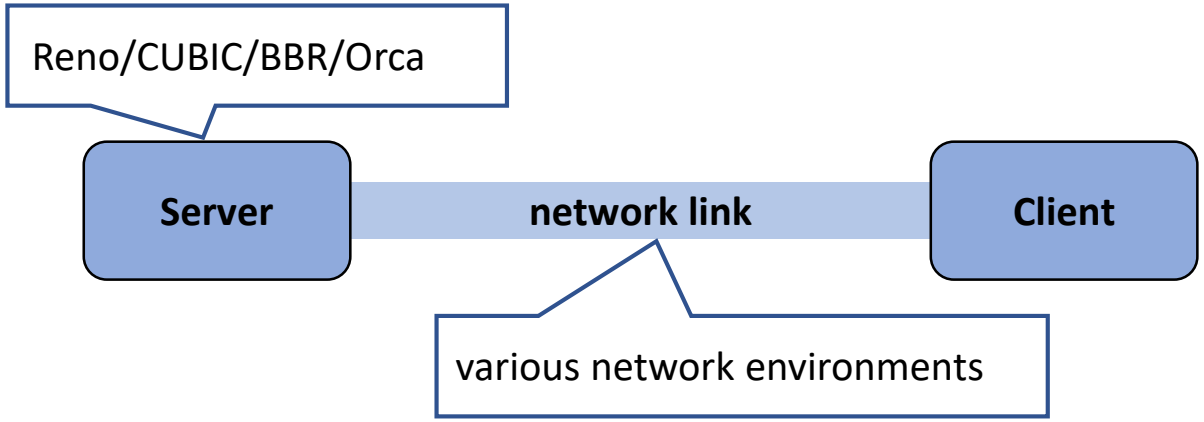
Highly dynamic network environment:
5G, Wi-Fi, statellite, optical fiber, etc.



Many congestion control (CC) algorithms have been proposed for various network environments:
Reno, CUBIC, BBR, Orca, etc.

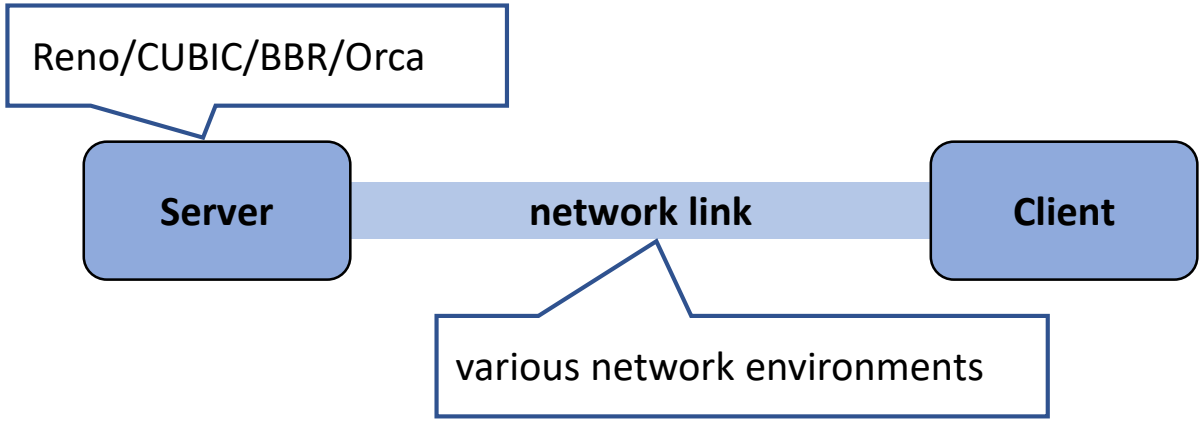
Characterize CC algorithm behaviour

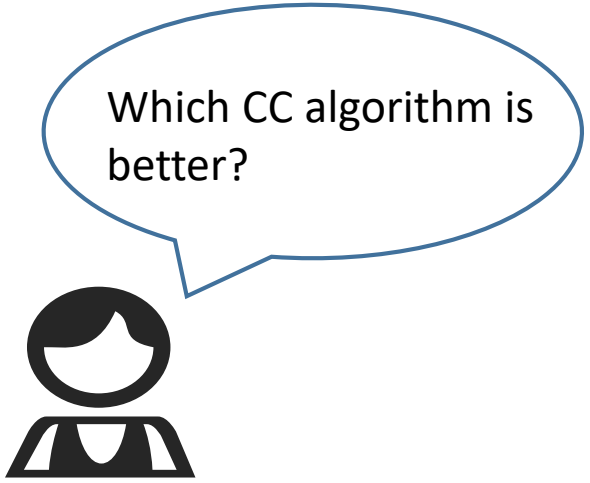
Simulator	Intelligence Enabled	Language	GUI Support	Ease of Use	Available Network
Ours	✓	C++ and Python	✗	Easy	Wired, Wireless, Mobile Networks
OMNeT++	✗	C++	✓	Easy	Wired, Wireless, Adhoc and WSN
ns-3	✗	C++ and Python	✓	Hard	Wired, Wireless, Adhoc and WSN
OPNET	✗	C and C++	✓	Easy	Wired, Wireless, Adhoc and WSN
NetSim	✗	C and Java	✓	Easy	Wired&Wireless Sensor Networks



Characterize CC algorithm behaviour

Simulator	Intelligence Enabled	Language	GUI Support	Ease of Use	Available Network
Ours	✓	C++ and Python	✗	Easy	Wired, Wireless, Mobile Networks
OMNeT++	✗	C++	✓	Easy	Wired, Wireless, Adhoc and WSN
ns-3	✗	C++ and Python	✓	Hard	Wired, Wireless, Adhoc and WSN
OPNET	✗	C and C++	✓	Easy	Wired, Wireless, Adhoc and WSN
NetSim	✗	C and Java	✓	Easy	Wired&Wireless Sensor Networks





Learning-based CC simulator

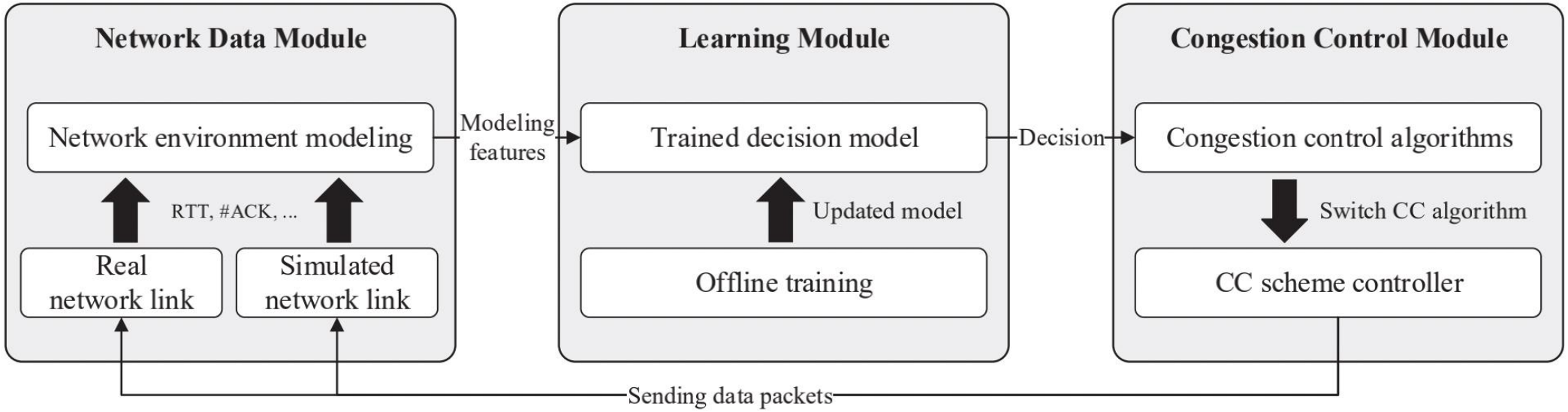
Key idea:
utilizing learning-based methods (AI and ML technologies) to make accurate decisions on switching CC algorithms for TCP connections in different network environment.

Design goal:
choose the optimal CC scheme dynamically according to network environment

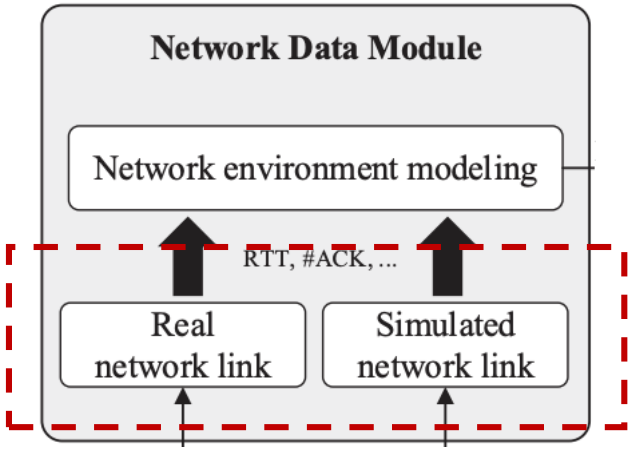
Learning-based CC simulator

Key idea:
utilizing learning-based methods (AI and ML technologies) to make accurate decisions on switching CC algorithms for TCP connections in different network environment.

Design goal:
choose the optimal CC scheme dynamically according to network environment

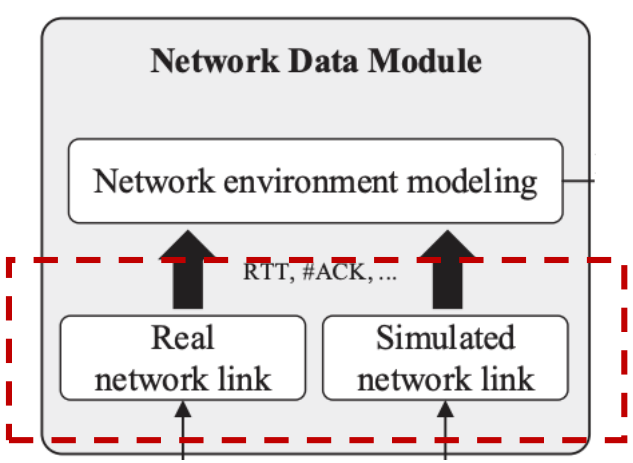


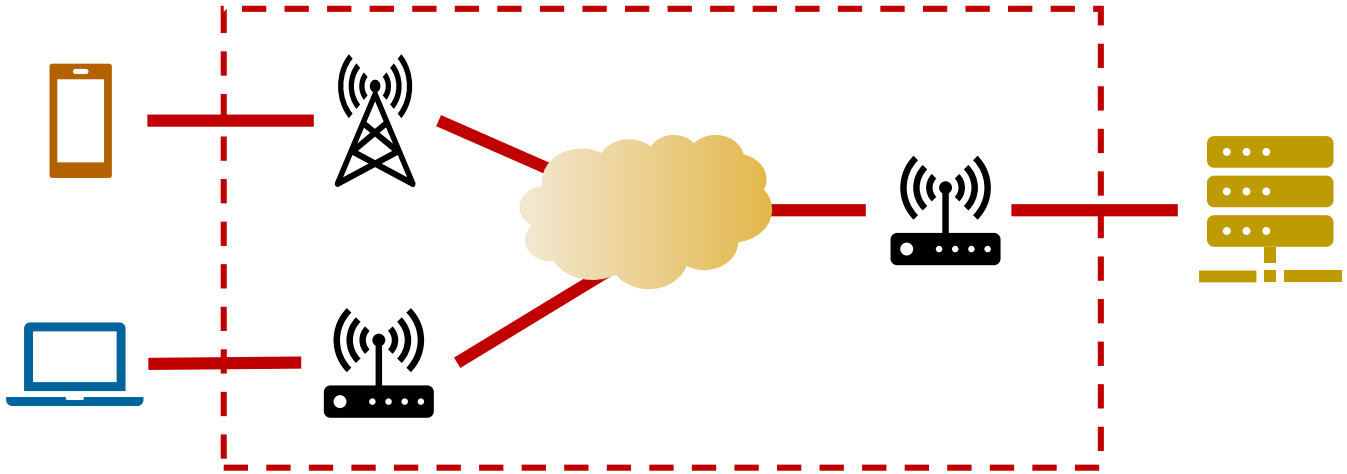
Network data module



Network data module

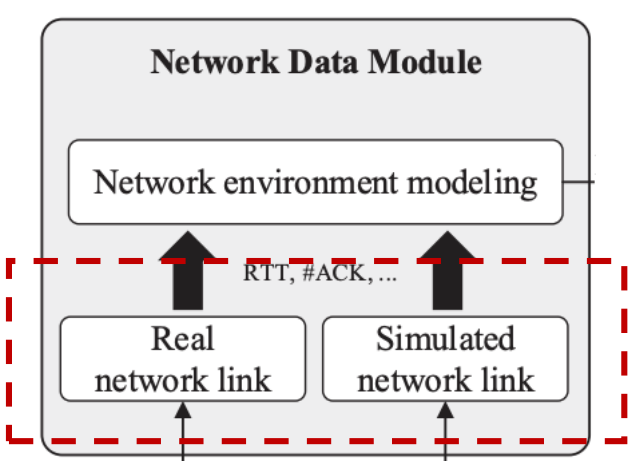
We abstract a complex data link as a bottleneck link $\delta \leftarrow (BtlThr, RTT, Lossrate)$

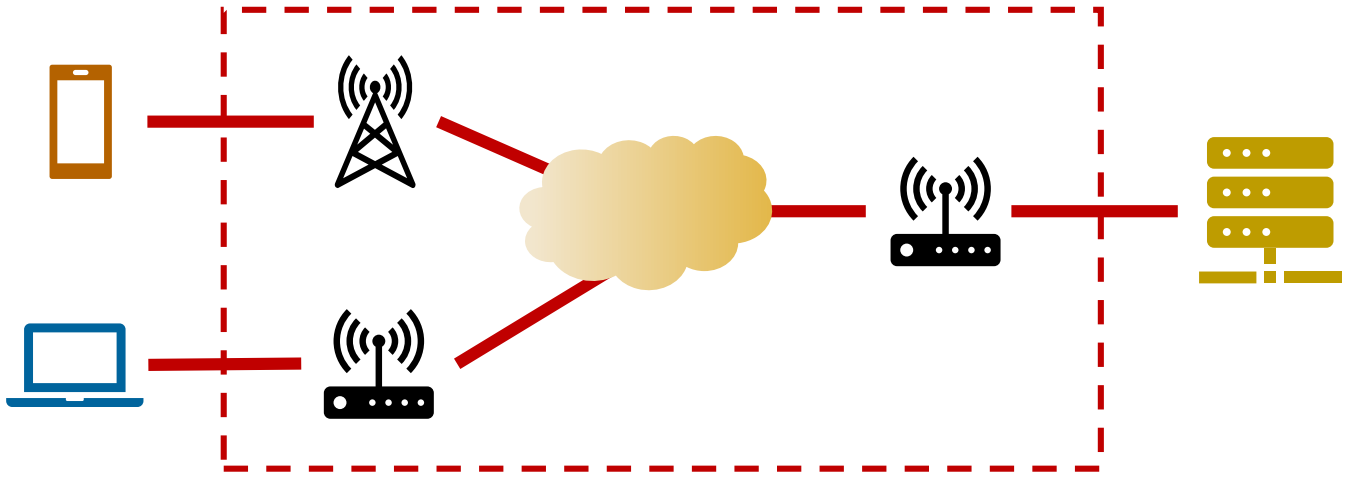




Network data module

We abstract a complex data link as a bottleneck link $\delta \leftarrow (BtlThr, RTT, Lossrate)$



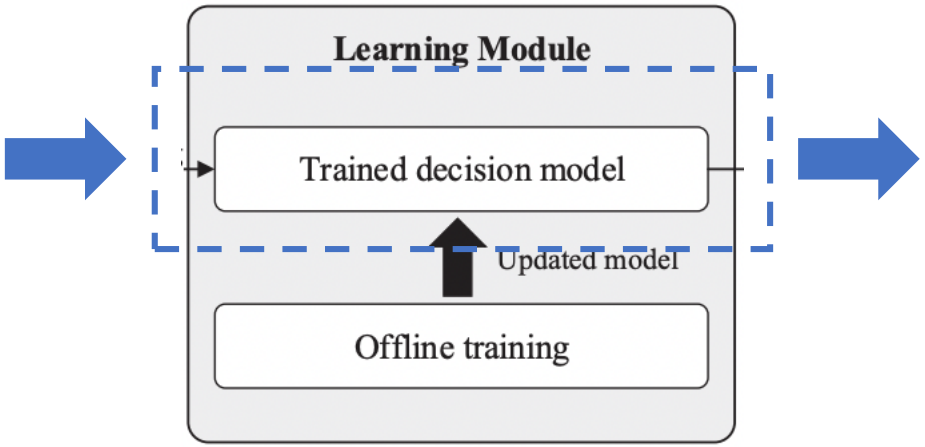


$$BtlThr = \min(Thr_1, Thr_2, \cdots, Thr_n), \quad RTT = \sum_{i=1}^{n} RTT_i, \quad Lossrate = 1 - \sum_{i=1}^{n}(1 - Lossrate_i).$$

Learning module

The bottleneck link triple

$\delta \leftarrow (BtlThr, RTT, Lossrate)$



Decision of CC algorithms selection

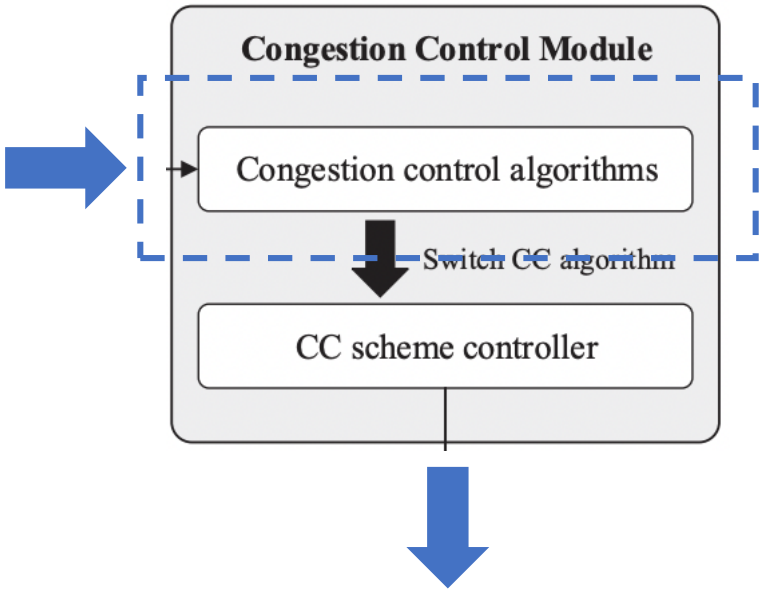
A mapping from input space to output space (classification problem)

Decision tree learning, SVM, CNN,

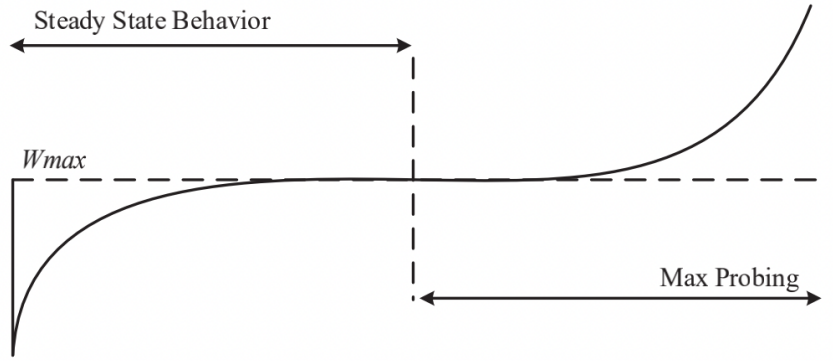
Random forest

Congestion control module

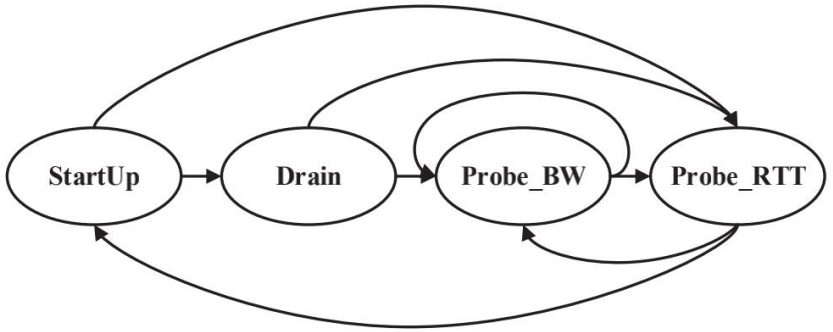
Decision of CC algorithms selection



Call kernel function to switch to the new CC scheme



The congestion window growth function of CUBIC.



The working state machine of BBR.

The set of candidate CC algorithms

Evaluation: offline training

- These parameters are combined to form 880 different triples, i.e., 880 types of network environments.

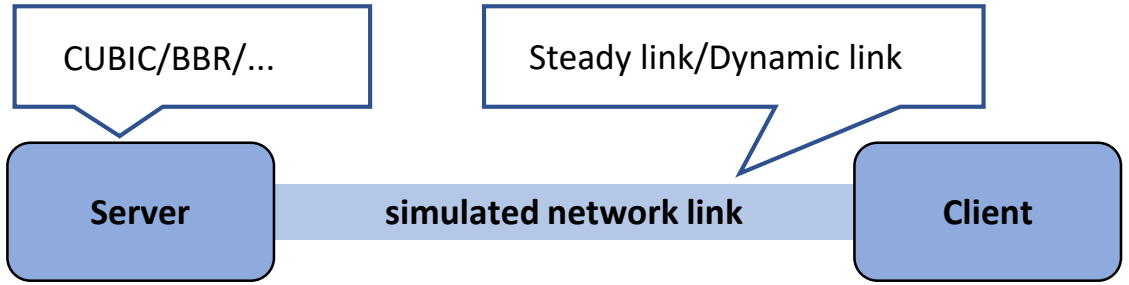
Optional data link parameters for generating training datasets

Parameter	Value
BltThr	0.6, 1.2, 2, 4, 6, 8, 10, 12,16, 20, 24 (Mbps)
RTT	30, 50, 75, 100, 150, 200, 250, 300 (ms)
Lossrate	0, 0.05, 0.1, 0.3, 0.5, 1, 3, 5, 10, 20 (%)

- In each network environment, we evaluate the performance of CUBIC and BBR respectively, totally for 1760 experiments.
- We select the CC algorithm which has a larger throughput in the network environment as the label of this network scenario.
- In order to improve the generalization of the trained model and avoid over-fitting, we utilize the EL method to aggregate 10 independent DT learning.

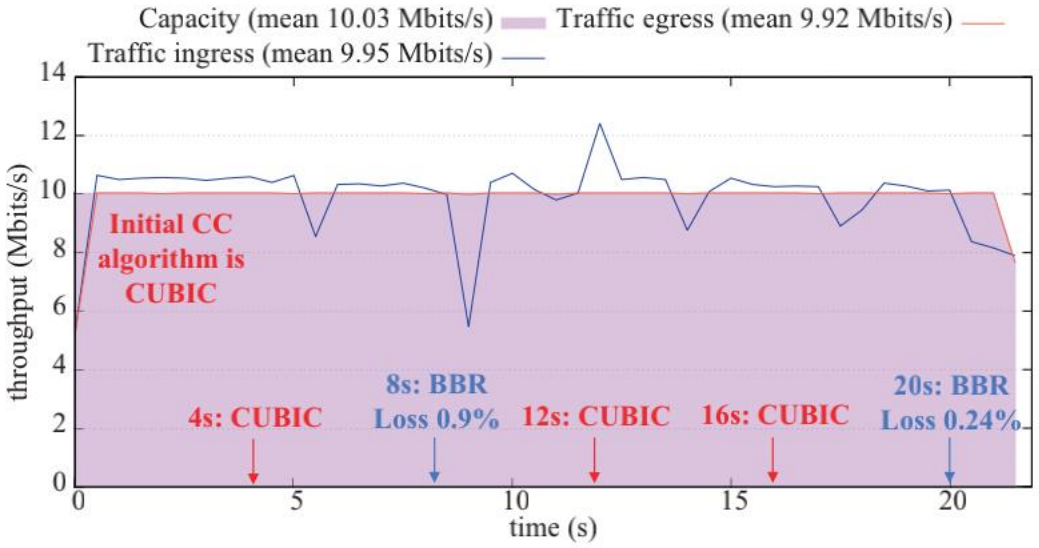
Evaluation: network link simulation

- In the network data module, we used Mahimahi to implement the network link, which can adjust *(BltThr, RTT, Lossrate)* to simulate different network links.

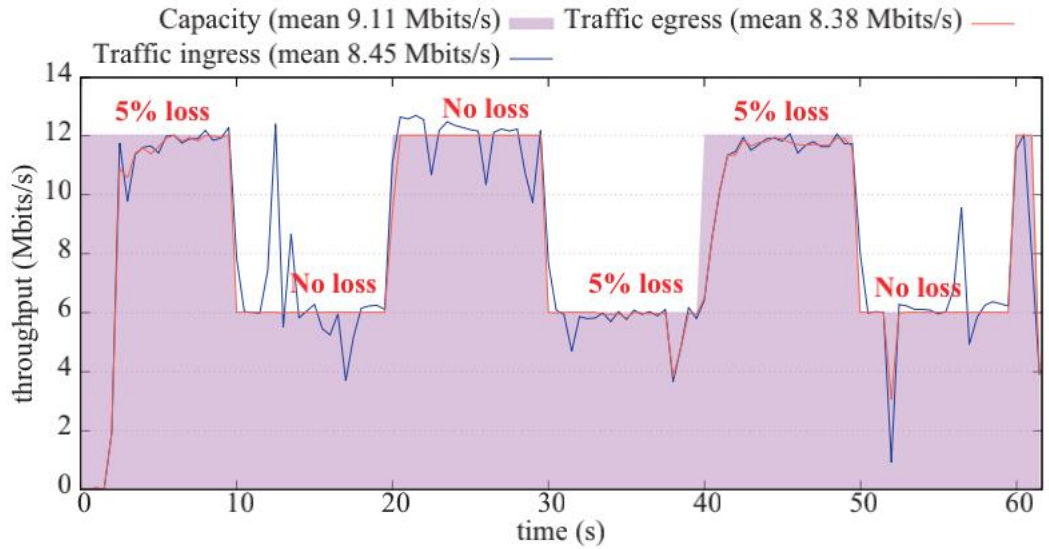


- **Steady link:** (50ms, 10mbps, 0%), last 20s
- **Dynamic link:** (200ms, 12mbps, 5%) -> (200ms, 6mbps, 0%)
  -> (200ms, 12mbps, 0%) -> (200ms, 6mbps, 5%), change state every 10s and last 60s
- The decision interval of simulator is 4s

Evaluation: result



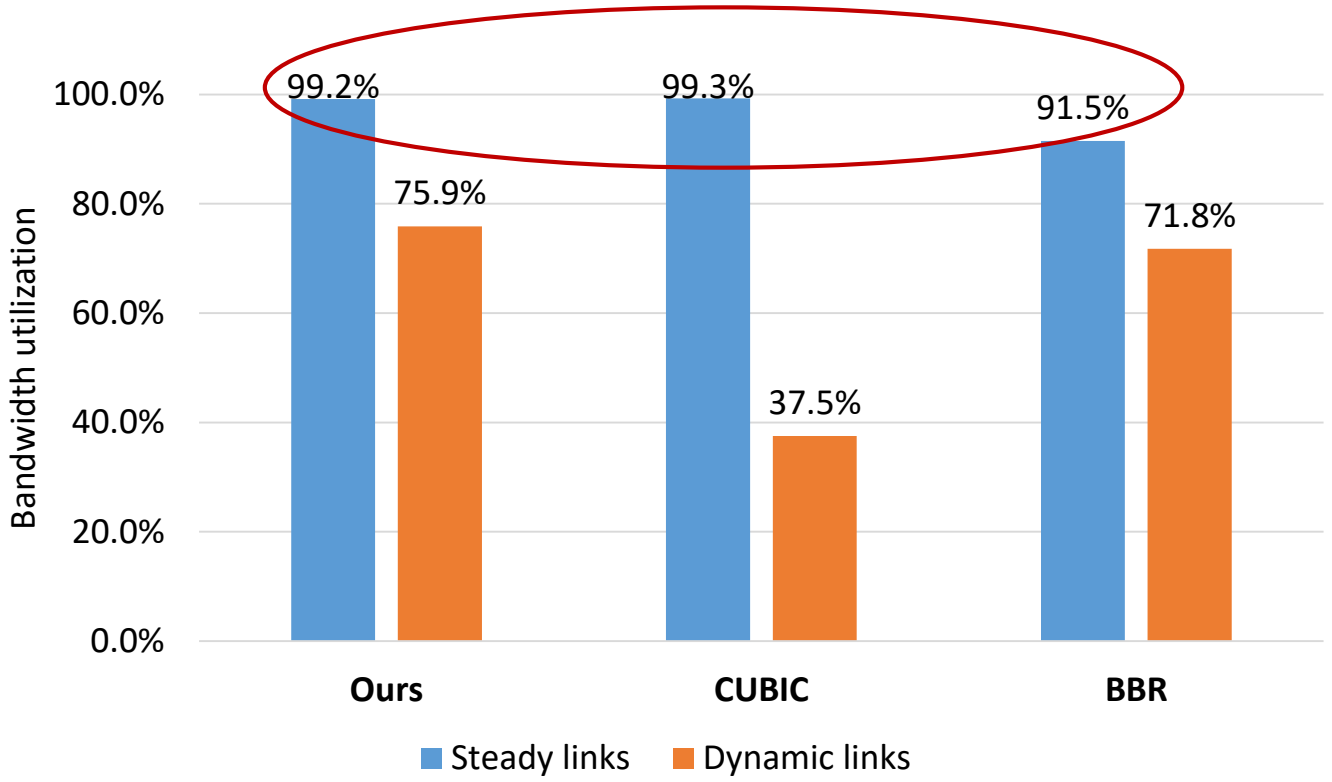
The simulator performance in steady links.
(50ms, 10mbps, 0%)



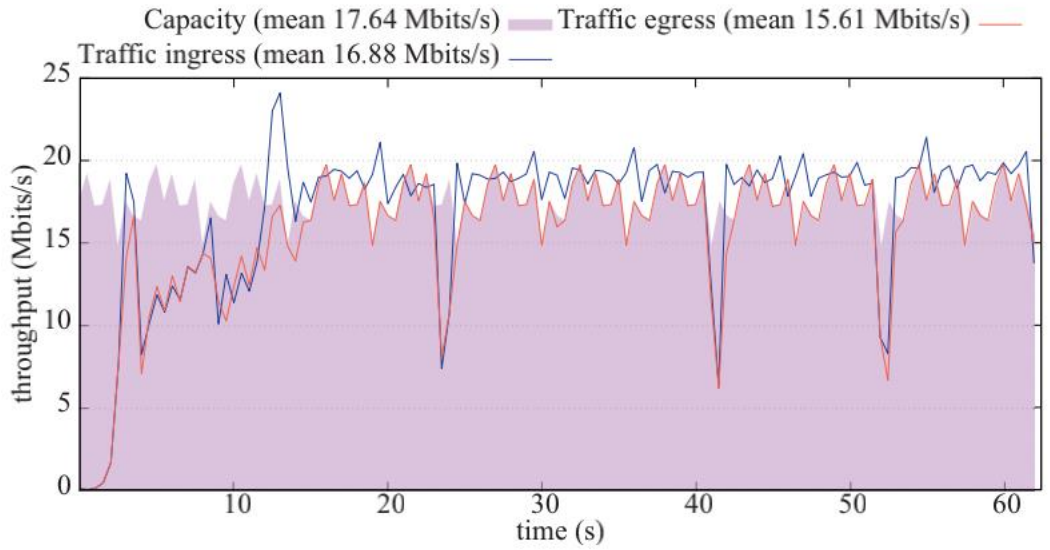
The simulator performance in dynamic links.
(200ms, 12mbps, 5%) -> (200ms, 6mbps, 0%) -> (200ms, 12mbps, 0%) -> (200ms, 6mbps, 5%)

Evaluation: result

- Our simulator always maintains optimal or sub-optimal bandwidth utilization in both steady links and dynamic links.
- The performance of our simulator is very close to the optimal CC algorithm when it is suboptimal (within 1%).
- Our simulator maintains a clear gap between the performance of the worst CC algorithm (above 7%).



Evaluation: result



The simulator performance in real world links. *The real network traffic datasets are sampled from an online audio and video site.*

Conclusion & Thank you!

Pros.

- ✓ Enable learning methodologies in the network simulator for fined decision-making on the selection of CC algorithms
- ✓ Provide students an intuitive feeling that which CC algorithm performs better under specific network environments
- ✓ High flexibility, availability and scalability

Future work

- ➢ Enrich the test link environment and refine the modeling of the network environment.
- ➢ Embed more CC algorithms and learning methods in the simulator as default options
- ➢ Add a visualization interface for the simulator to provide easier operations