L4S in Practice: Assessing the Feasibility of Incremental Deployment

Fatih Berkay Sarpkaya, Ashutosh Srivastava, Fraida Fund, Shivendra Panwar

KNIT 8: A FABRIC Community Workshop - March 20, 2024

NYU | TANDON SCHOOL OF ENGINEERING

Outline

- Introduction
  - Background for L4S
  - L4S Deployment Requirements
  - Problem Statement
- How does FABRIC support this research?
  - Single Bottleneck Experiments
  - Multiple Bottleneck Experiments
- Initial Findings / Results
- Conclusion

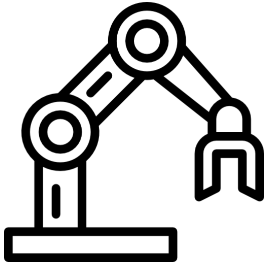
Outline

- Introduction
  - Background for L4S
  - L4S Deployment Requirements
  - Problem Statement
- How does FABRIC support this research?
  - Single Bottleneck Experiments
  - Multiple Bottleneck Experiments
- Initial Findings / Results
- Conclusion

Introduction

Interactive high-rate latency-critical applications are becoming more and more widespread.

- Online Gaming
- Virtual Reality (VR)
- Remote Control



L4S[1]

→ **L**ow **L**atency

→ **L**ow **L**oss

→ **S**calable Throughput

(Broadly supported by the cable industry for "10G" networking)[2]

1 B. Briscoe, K. De Schepper, M. Bagnulo, and G. White, "RFC 9330: Low latency, low loss, and scalable throughput (L4S) internet service: Architecture," USA, 2023.

2 https://www.cablelabs.com/blog/l4s-interop-lays-groundwork-for-10g-metaverse

L4S: Low Latency, Low Loss and Scalable Throughput

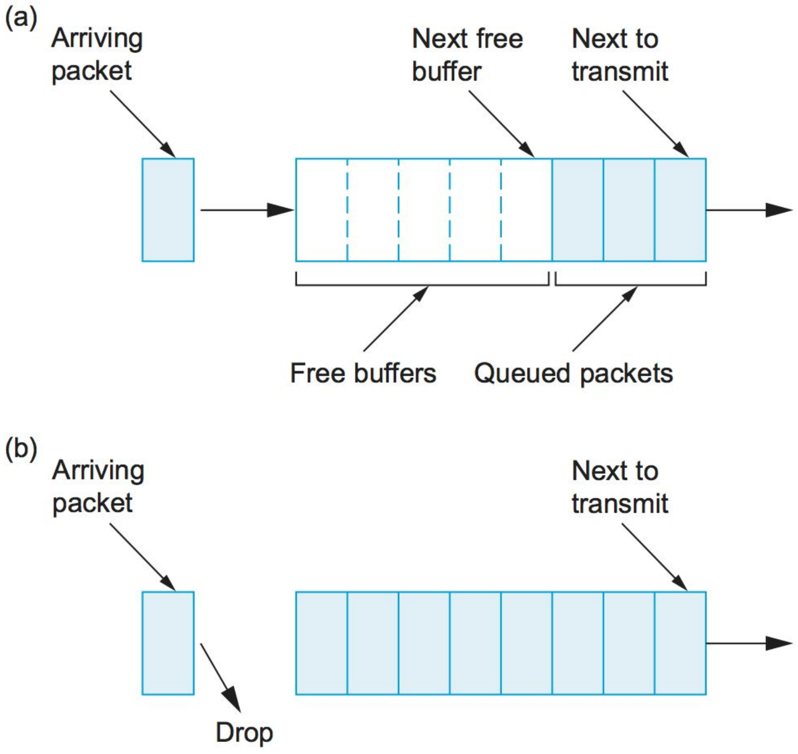
Core Mechanisms:

- Scalable Congestion Control
- Accurate Explicit Congestion Notification (AccECN)
- Dual Queue Active Queue Management (AQM)

Not like other mechanisms for low latency - involves endpoints *and* middleboxes.

Pre-L4S: Classic Behavior of Buffers

- Tail drop increases the Queuing delay!



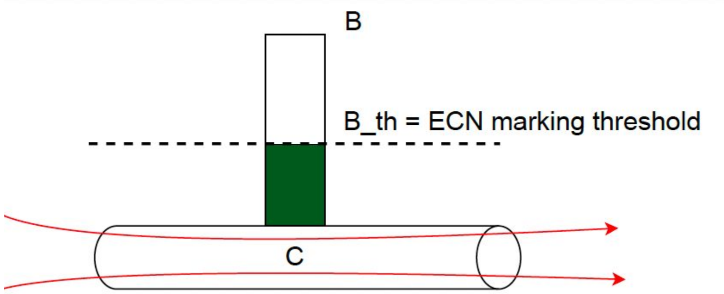
Source: Larry L. Peterson and Bruce S. Davie. 2011. Computer Networks, Fifth Edition: A Systems Approach (5th. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pre-L4S: Explicit Congestion Notification (ECN)[3]

- The network (bottleneck router) can signal congestion without having to drop packets!
- Use 2 bits in the IP header for ECN marking.

Idea:

- If queue size at router crosses a threshold, mark the packet as "congestion experienced" (CE).
- The TCP receiver can feedback this signal to the sender via the ACKs (using 1 TCP header bit).



Requirement:

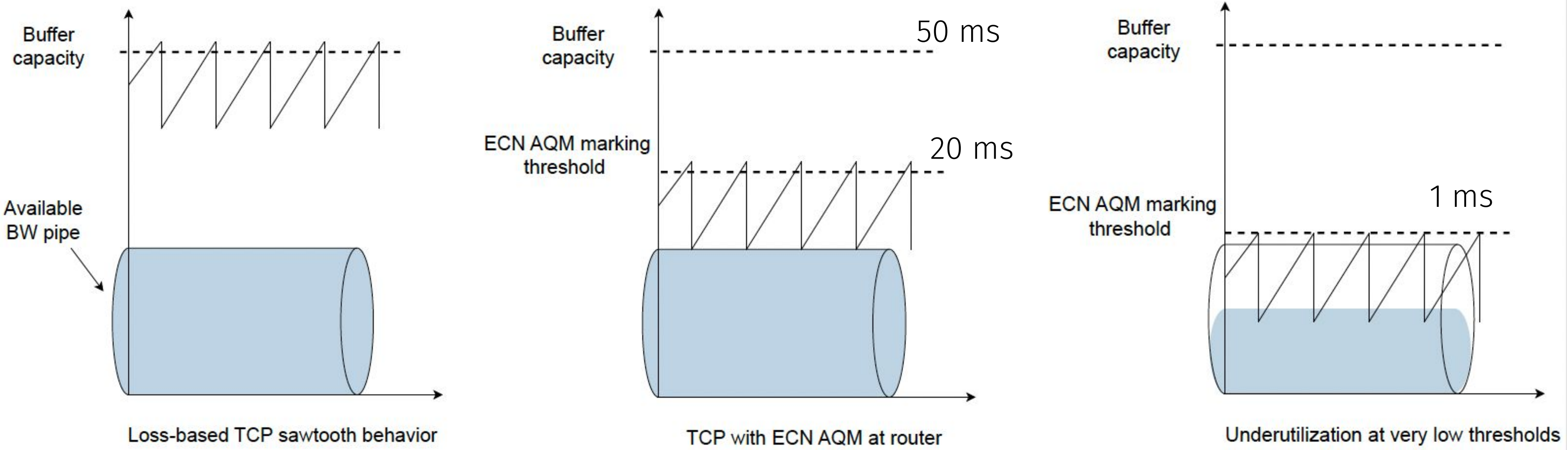
- Network nodes have to support ECN.

No need to wait for the buffer to fill up to observe congestion -> lower queuing delay

[3] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc316

Pre-L4S: TCP congestion control + ECN

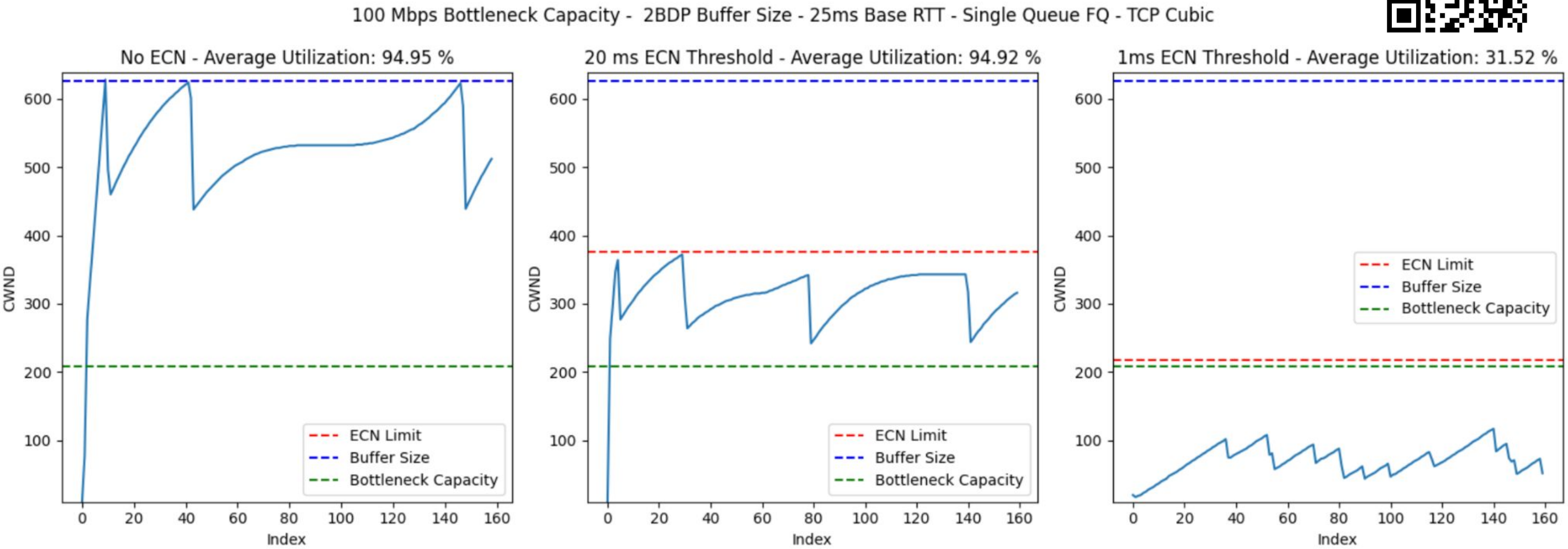
AQM : Active Queue Management (at routers)



Issue: Extremely low queuing delays are not achievable due to underutilization of capacity!

Source: B. Briscoe, K. D. Schepper, O. Albisser, O. Tilmans, N. Kuhn, G. Fairhurst, R. Scheffenegger, M. Abrahamsson, I. Johansson, P. Balasubramanian, D. Pullen, G. Bracha, J. Morton, and D. Taht, "Implementing the 'prague requirements 'for low latency low loss scalable throughput (l4s)," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:180259180

Pre-L4S: TCP congestion control + ECN



Loss-based TCP sawtooth behavior

TCP with ECN AQM at router

Underutilization at very low thresholds

Issue: Extremely low queuing delays are not achievable due to underutilization of capacity!

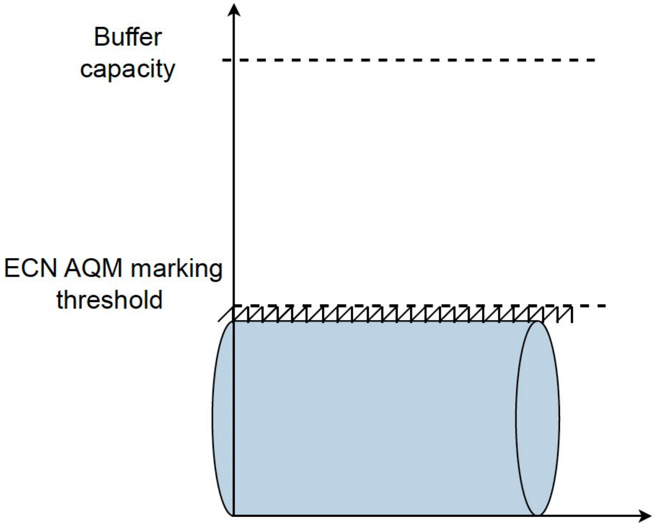
The experiment is available here: https://github.com/fatihsarpkaya/TCP-ECN

Scalable Congestion Control, AccECN

Key idea

Instead of reducing CWND by half, react in proportion to the extent of congestion

ECN Marks	Legacy TCP	Scalable TCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%



L4S CC- small sawtooth -> Full utilization

Requires Accurate Explicit Congestion Notification (AccECN)

100% utilization and extremely low queueing delay is possible with L4S

Source: B. Briscoe, K. D. Schepper, O. Albisser, O. Tilmans, N. Kuhn, G. Fairhurst, R. Scheffenegger, M. Abrahamsson, I. Johansson, P. Balasubramanian, D. Pullen, G. Bracha, J. Morton, and D. Taht, “Implementing the ’prague requirements ’for low latency low loss scalable throughput (l4s),” 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:180259180

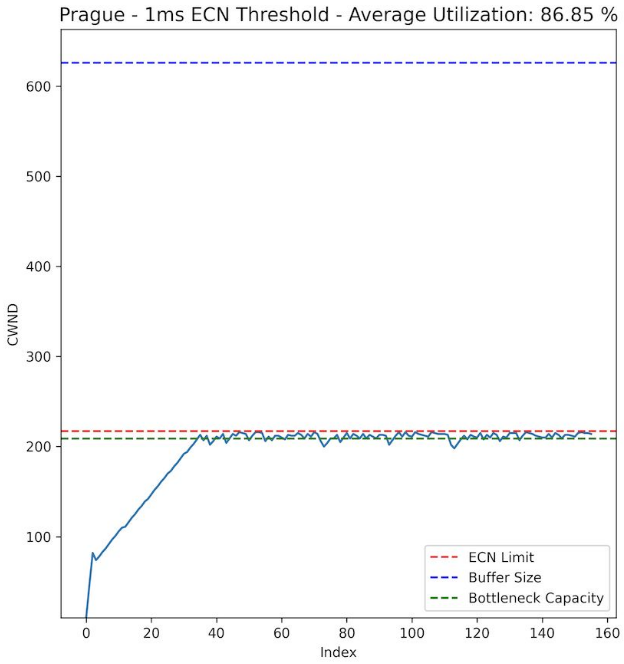
Scalable Congestion Control, AccECN

Key idea

Instead of reducing CWND by half, react in proportion to the extent of congestion

ECN Marks	Legacy TCP	Scalable TCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

Requires Accurate Explicit Congestion Notification (AccECN)



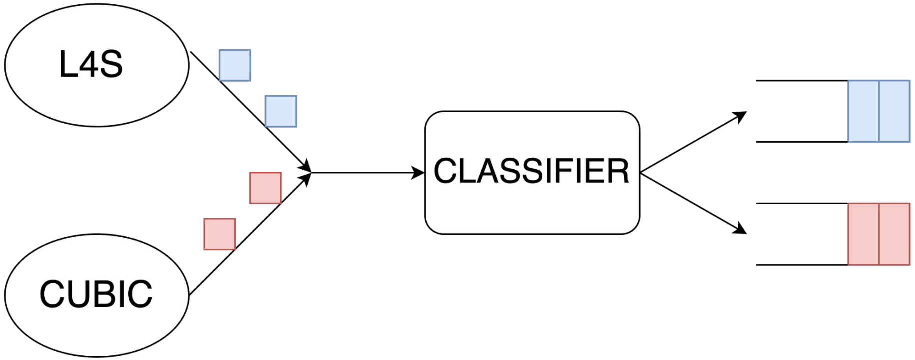
Very high utilization and extremely low queueing delay is possible with L4S

Source: B. Briscoe, K. D. Schepper, O. Albisser, O. Tilmans, N. Kuhn, G. Fairhurst, R. Scheffenegger, M. Abrahamsson, I. Johansson, P. Balasubramanian, D. Pullen, G. Bracha, J. Morton, and D. Taht, "Implementing the 'prague requirements 'for low latency low loss scalable throughput (l4s)," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:180259180

Dual Queue AQM[4]

L4S **does not coexist well** with classic TCP over a shared bottleneck. Fine sawtooth TCP (scalable TCP) dominates large sawtooth TCP (CUBIC) when they share a bottleneck with a fixed ECN threshold.

Idea: separate scalable flows and "classic" flows into separate queues.

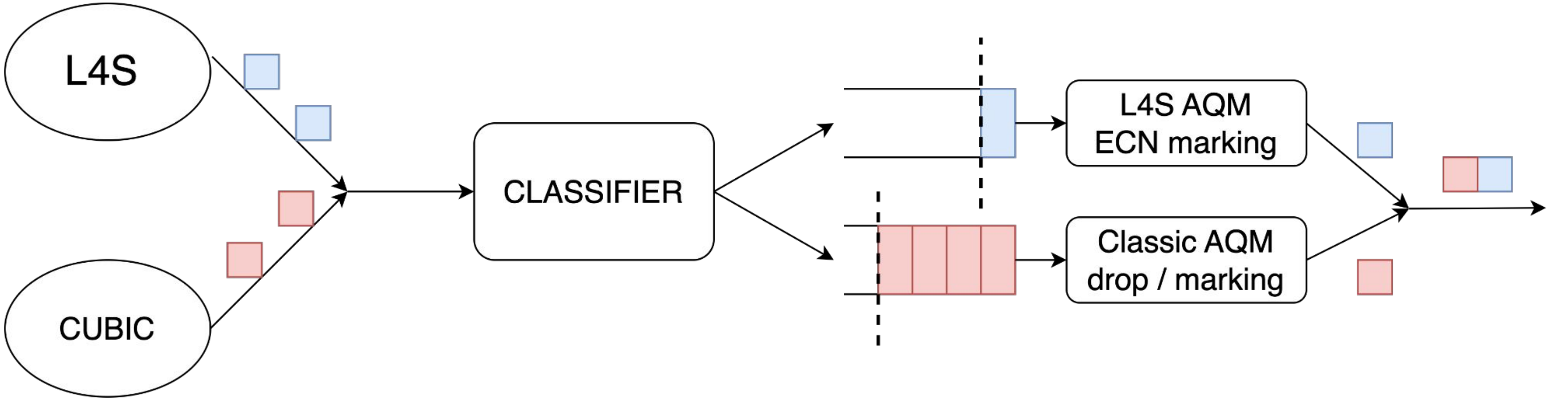


[4] O. Albisser, K. De Schepper, B. Briscoe, O. Tilmans, and H. Steen, "DUALPI2—Low Latency, Low Loss and Scalable (L4S) AQM," Net-Dev 0x13, Prague, 2019.

Dual Queue AQM

Routers cannot use low ECN thresholds because they would be harmful to "classic" TCP flows! But scalable flows prefer low thresholds.

Idea: the two queues should use different marking thresholds for "classic" and scalable flows



Dual Queue AQM

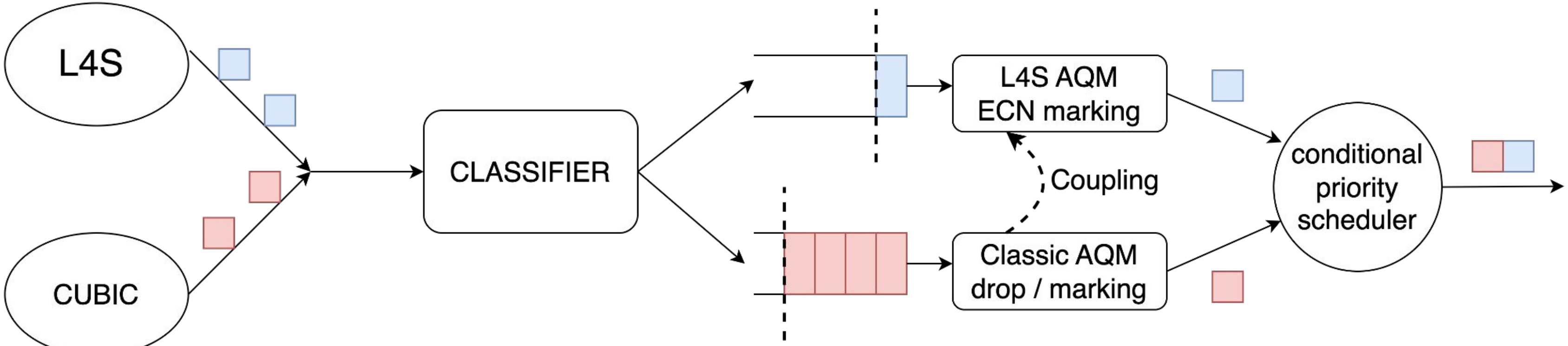


Fig: Dual Queue Coupled AQM

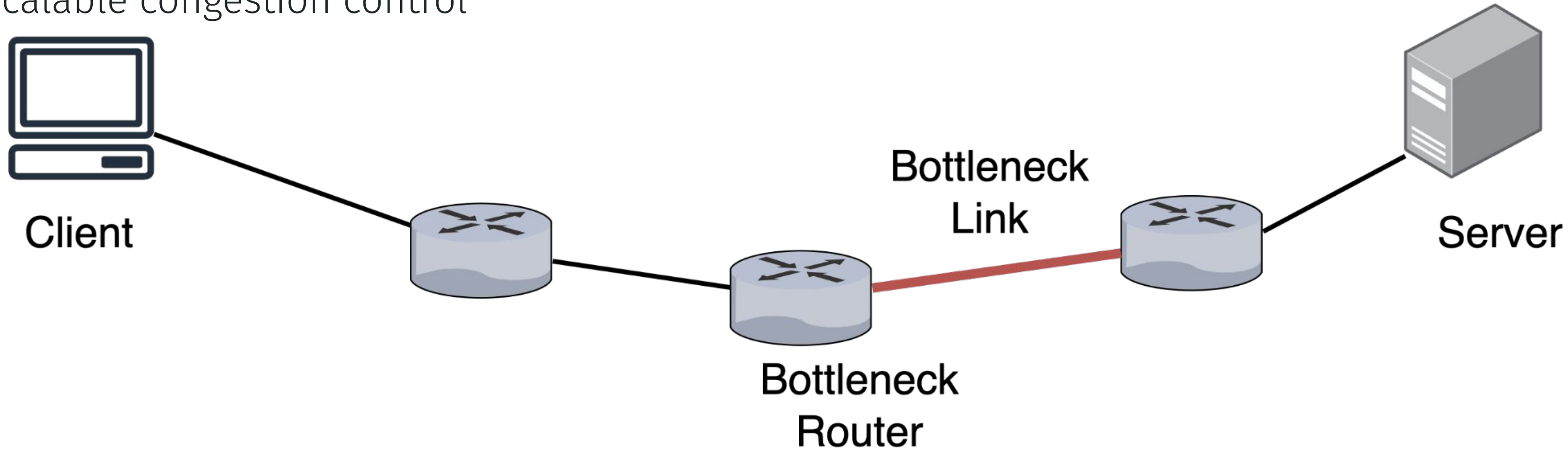
Binary Codepoint	Codepoint Name	Meaning
00	Not-ECT	Not ECN-capable transport
01	ECT(1)	L4S-capable transport
10	ECT(0)	Not L4S-capable transport
11	CE	Congestion Experienced

Table I : L4S codepoints and meaning

L4S Requirements Summary

AccECN support
Scalable congestion control

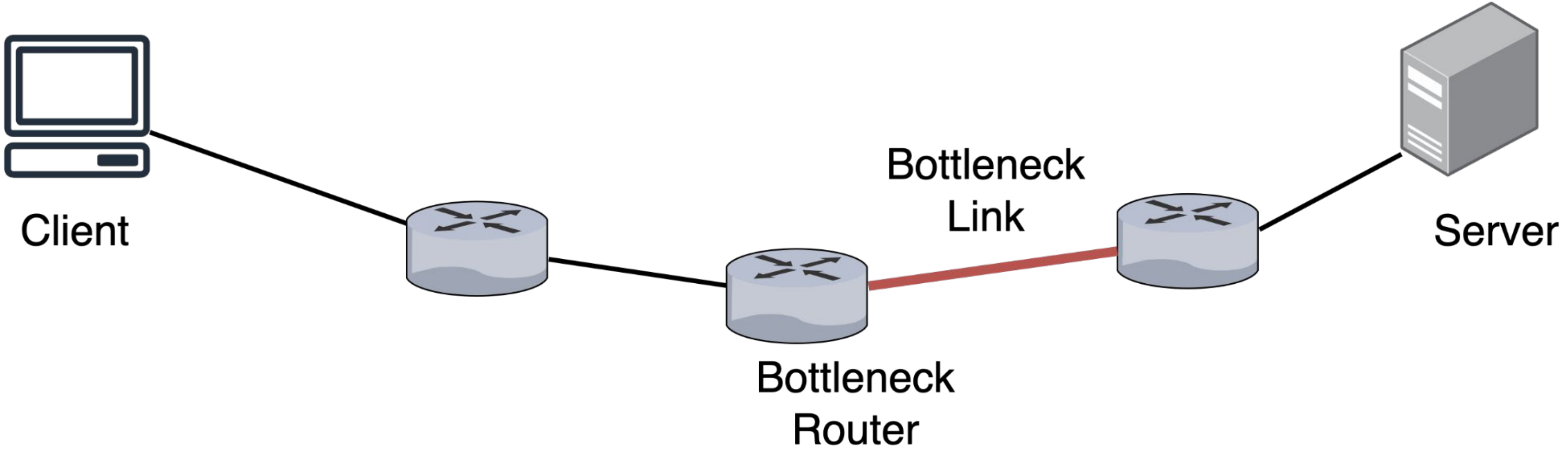
AccECN support



Separate queues
ECN marking
Different ECN thresholds

L4S Requirements Summary

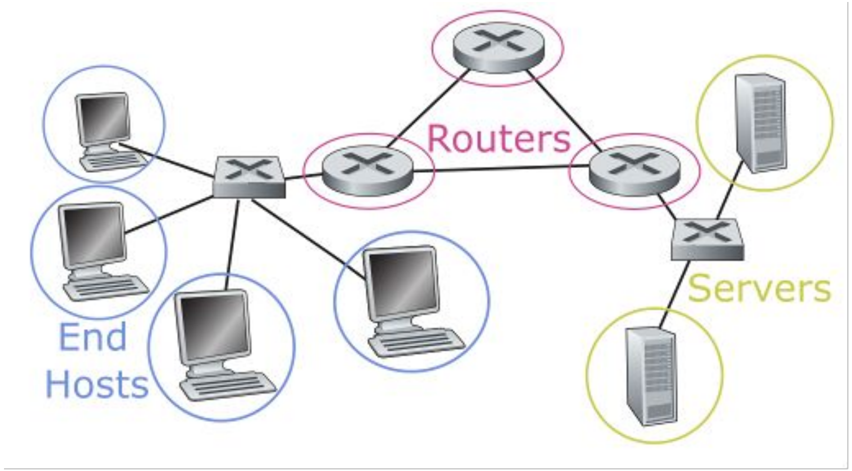
What if any one endpoint or router does not support L4S requirements?



A new protocol won't be deployed universally, all at once - there will be some incremental deployment

L4S ideal: widespread changes to Internet hosts, servers, and routers to reduce latency.

L4S practical: incremental deployment: ??

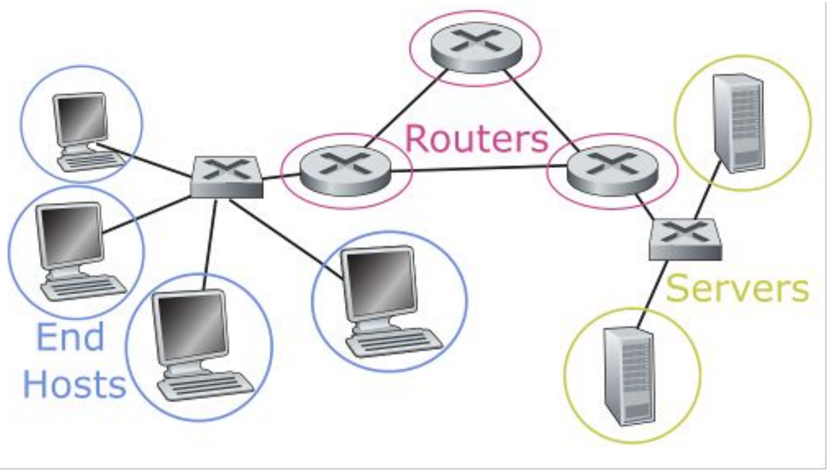


If "incremental deployment" is worse than the status quo, it will be difficult to *ever* reach widespread deployment.

Research question

What would the Internet look like with a partial L4S deployment?

Would a partial deployment encourage further deployment, or encourage reversion to status quo?

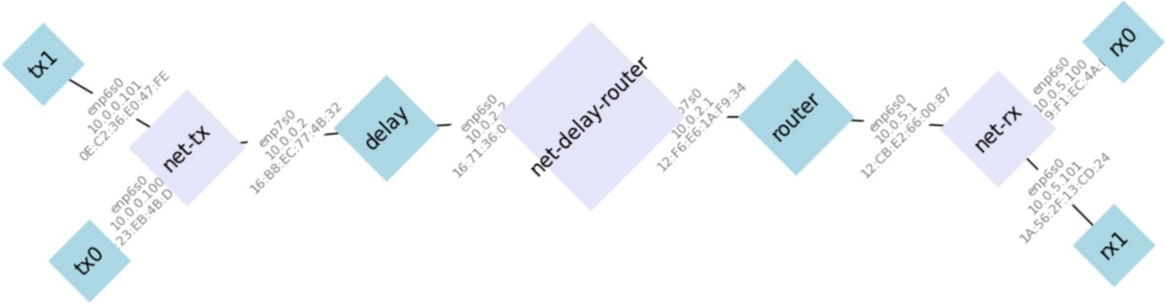


Outline

- Introduction
  - Background for L4S
  - L4S deployment requirements
  - Problem Statement
- How does FABRIC support this research?
  - Single bottleneck experiments
  - Multiple bottleneck experiments
- Initial Findings / Results
- Conclusion

Single bottleneck experiment

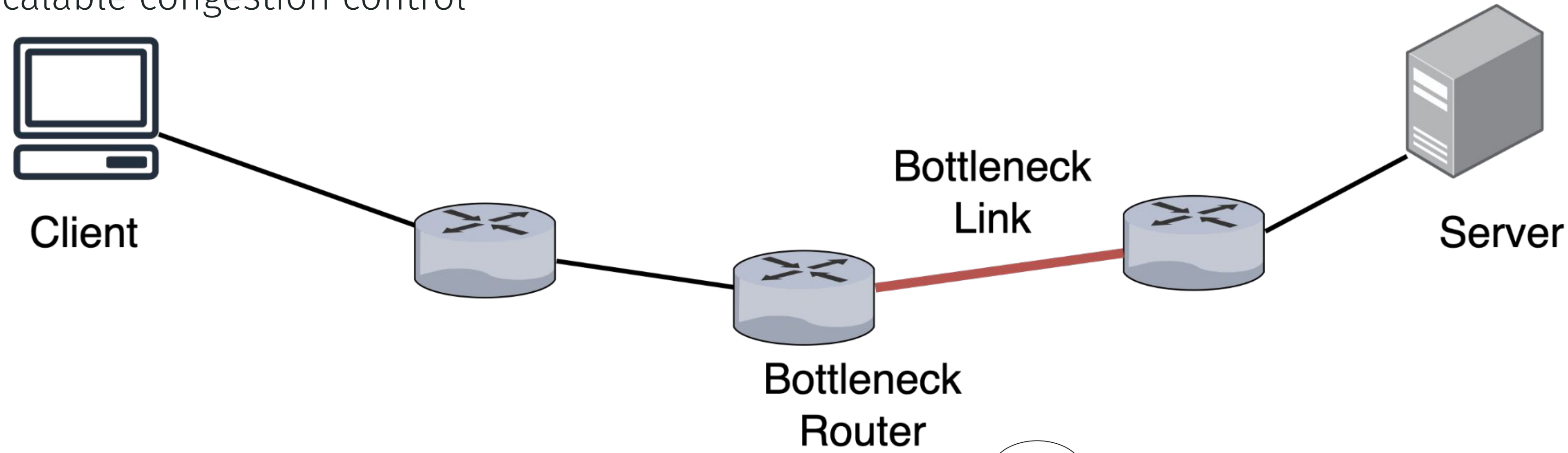
- Topology as illustrated below - two senders, two receivers, shared bottleneck link
- Want to understand behavior across a variety of network settings: buffer size, bottleneck capacity, propagation delay
- And for each network setting, different levels of L4S support
- Challenge: managing full-factorial experiment design



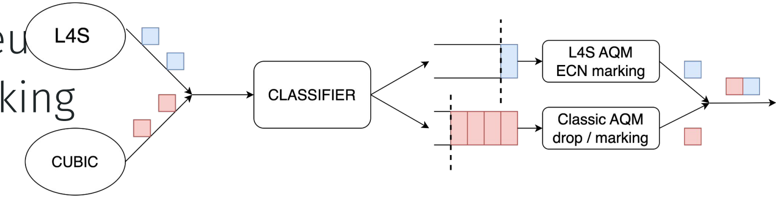
"Full" deployment

AccECN support
Scalable congestion control

AccECN support



Separate queu
ECN marking
Different ECN thresholds

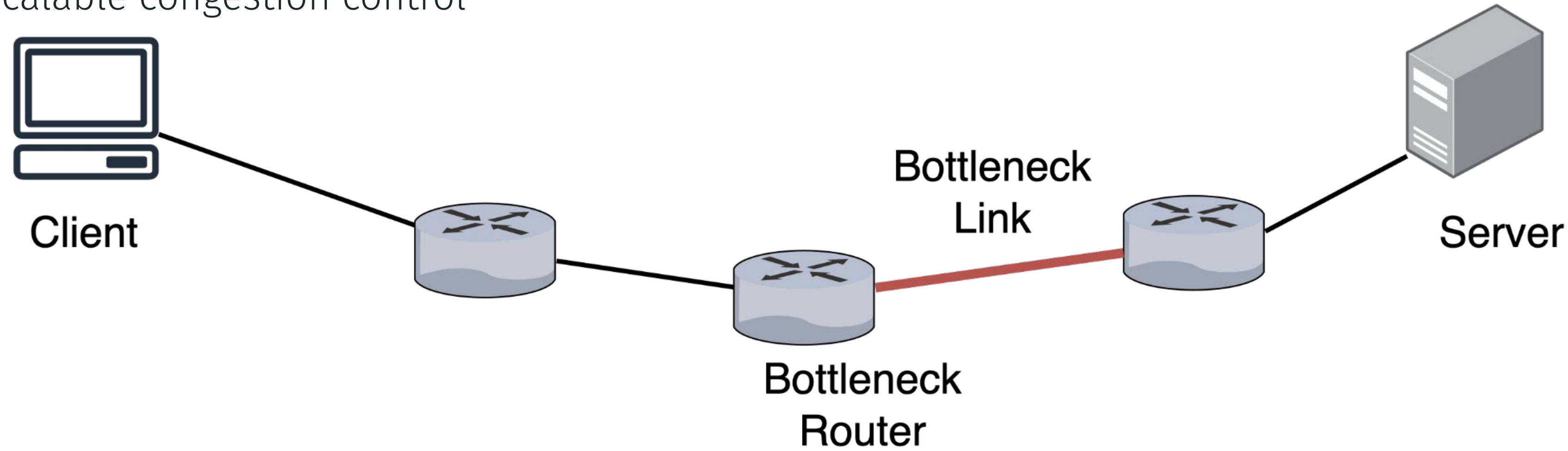


What if a bottleneck has a single queue & does not support ECN

AccECN support

Scalable congestion control

AccECN support



~~Separate queues~~

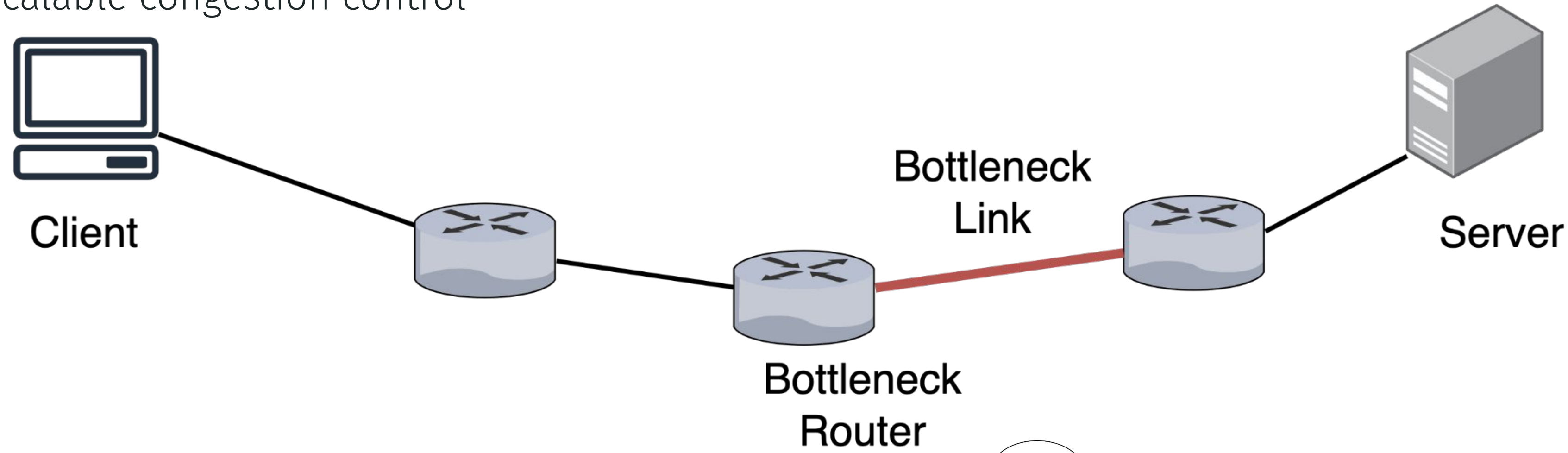
~~ECN marking~~

~~Different ECN thresholds~~

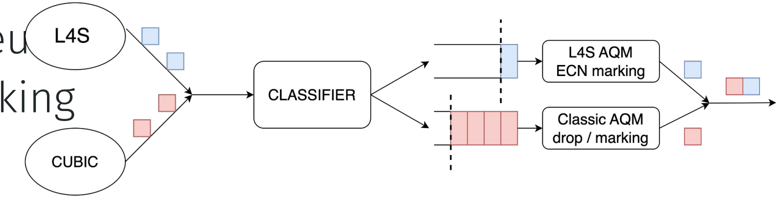
What if a bottleneck has fair queueing but one of the endpoints does not support ECN

AccECN support
Scalable congestion control

~~AccECN support~~



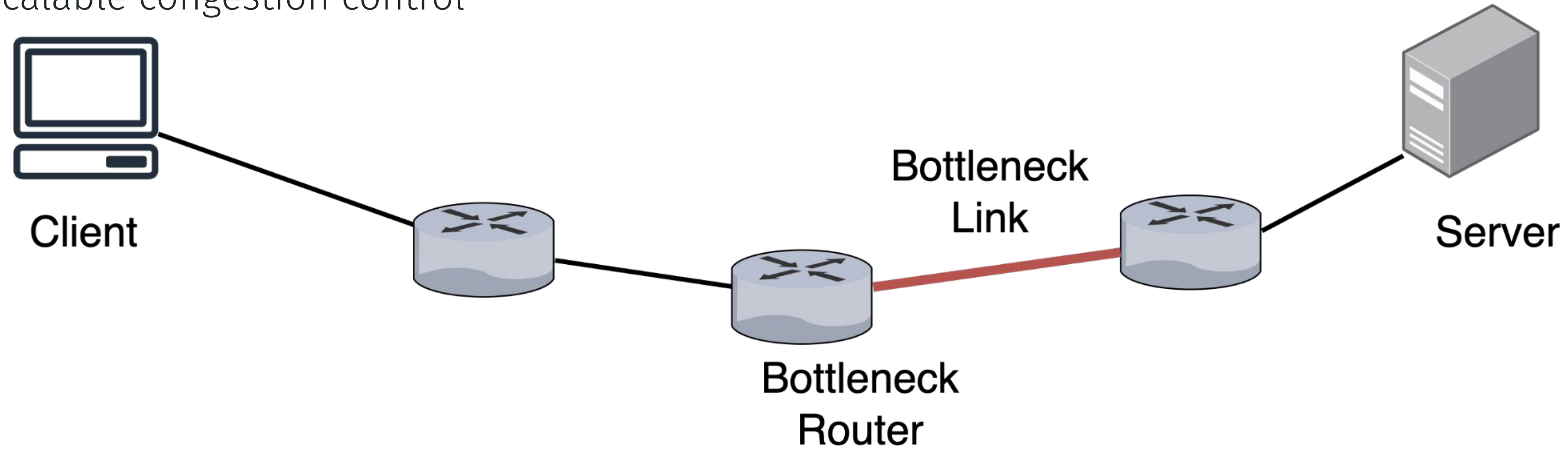
Separate queu
ECN marking
Different ECN thresholds



What if the bottleneck supports classic ECN but not L4S?

AccECN support
Scalable congestion control

AccECN support

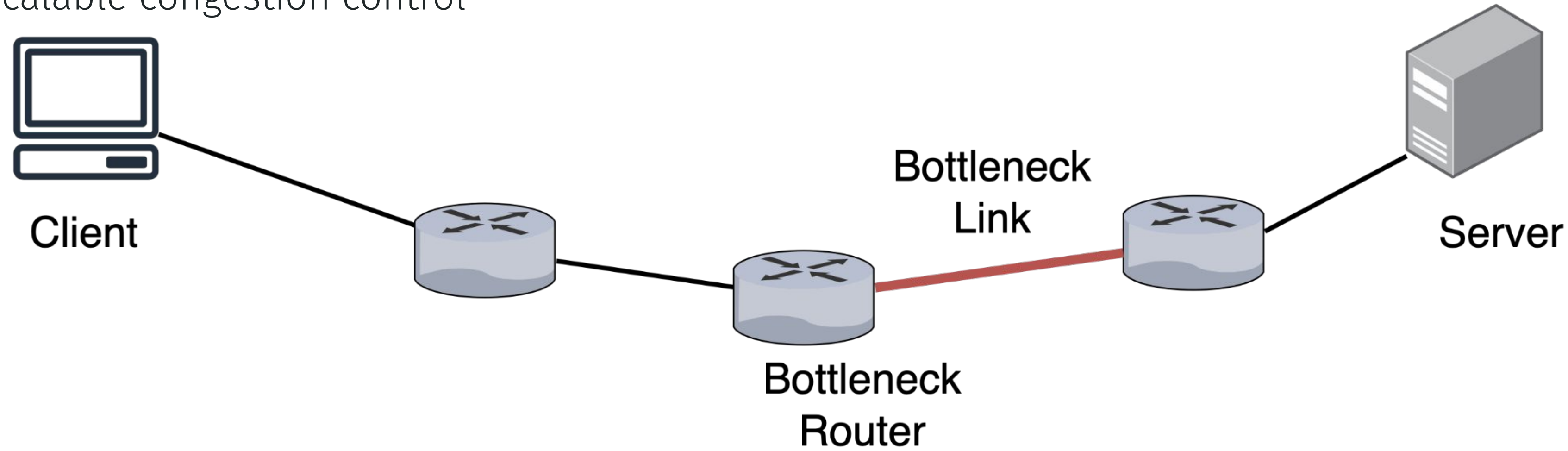


~~Separate queues~~
ECN marking
~~Different ECN thresholds~~

What if there is no Dual Queue AQM but per-flow queuing?

AccECN support
Scalable congestion control

AccECN support



Separate queues
ECN marking
~~Different ECN thresholds~~

Experiment Factors

FACTOR	VALUES
Buffer Size (n x BDP)	[0.5, 2, 5, 10]
Bottleneck Capacity (Mbps)	[**100**, 1000]
Base RTT (ms)	[**5, 10, 50**, 100]
AQM Type	[FIFO, Single Queue FQ, CoDel, FQ, FQ Codel, DualPI2]
ECN Threshold (ms)	[**1, 5**, 20]
ECN Fallback Algorithm	[**OFF**, ON]
Sender (L4S flow) ECN Support	[AccECN]
Sender (legacy flow) ECN Support	[Classic ECN]
Receiver (L4S flow) ECN Support	[No ECN, Classic ECN, **AccECN**]
Receiver (legacy flow) ECN Support	[No ECN, **Classic ECN**, AccECN]
Sender (L4S flow) Congestion Control	[TCP Prague]
Sender (legacy flow) Congestion Control	[TCP Cubic]
Number of Trials	5

Table II : Experiment Factors

Challenges and solutions

Full factorial design requires a very large number of unique experiments!

- Won't run all at once in a single Jupyter session
- May want to run different experiments on different slices in parallel
- Need to organize and keep track of all the experiment results

FABRIC's Jupyter notebook interface and Python library + some of our own tricks, makes this much easier to manage →

Generate list of full factorial experiments in notebook

```
import itertools

exp_factors = {
    'n_bdp': [0.5, 2, 5, 10],  # n x bandwidth delay product
    'btl_capacity': [100, 1000], #in Mbps #'btl_capacity': [100, 1000]
    'base_rtt': [5, 10, 50, 100], # in ms #'base_rtt': [5, 10, 50, 100],
    'aqm': ['FIFO', 'single_queue_FQ', 'Codel', 'FQ', 'FQ_Codel', 'DualPI2'],
    'ecn_threshold': [1,5,20], # in ms #'ecn_threshold': [1, 5, 20]
    'ecn_fallback': [0, 1],  #fallback algorithm, TCP Prague falls back to classic TCP when it detects single queue classic ECN bottleneck
    'rx0_ecn': [0,1,2],  # 0: noecn, 1: ecn, 2: accecn #'rx0_ecn': [0, 1, 2]
    'rx1_ecn': [0,1],  # 0: noecn, 1: ecn #'rx1_ecn': [0, 1]
    'cc_tx0': ["prague"],
    'cc_tx1': ["cubic"],
    'trial': [1,2,3,4,5] #'trial': [1, 2, 3, 4, 5]
}


factor_names = list(exp_factors.keys())
factor_lists = itertools.product(*exp_factors.values())

# Removing ECN factor from FIFO bottleneck because it does not support ECN
# Removing the cases where ECN Threshold is less than or equal to the buffer size in time, these cases are not meaningful in practice

exp_lists = [
    {k: v for k, v in zip(factor_names, fl)
     if k != 'ecn_threshold' or fl[factor_names.index('aqm')] != 'FIFO'}
    for fl in factor_lists
    if (fl[factor_names.index('n_bdp')] * fl[factor_names.index('base_rtt')] >= fl[factor_names.index('ecn_threshold')]
        or fl[factor_names.index('aqm')] == 'FIFO')
]



# Remove duplicates
exp_lists = [dict(t) for t in {frozenset(item.items()) for item in exp_lists}]
```

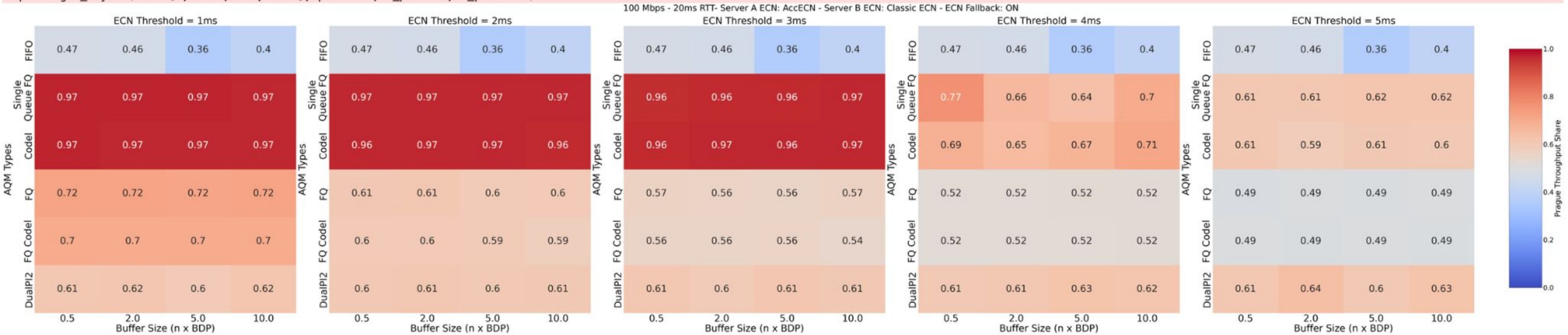
Allow stop/resume experiments in Jupyter

```
for exp in exp_lists:

    # check if we already ran this experiment
    # (allow stop/resume)
    name_tx0="%s_%0.1f_%d_%d_%s_%s_%d_%d_%d_%d" % (exp['cc_tx0'],exp['n_bdp'], exp['btl_capacity'], exp['base_rtt
    name_tx1="%s_%0.1f_%d_%d_%s_%s_%d_%d_%d_%d" % (exp['cc_tx1'],exp['n_bdp'], exp['btl_capacity'], exp['base_rtt

    file_out_tx0_json = name_tx0+"-result.json"
    file_out_tx0_ss = name_tx0+"-ss.txt"
    stdout_tx0_json, stderr_tx0_json = tx0_node.execute("ls " + file_out_tx0_json, quiet=True)
    stdout_tx0_ss, stderr_tx0_ss = tx0_node.execute("ls " + file_out_tx0_ss, quiet=True)

    file_out_tx1_json =name_tx1+"-result.json"
    file_out_tx1_ss = name_tx1+"-ss.txt"
    stdout_tx1_json, stderr_tx1_json = tx1_node.execute("ls " + file_out_tx1_json, quiet=True)
    stdout_tx1_ss, stderr_tx1_ss = tx1_node.execute("ls " + file_out_tx1_ss, quiet=True)

    if len(stdout_tx0_json) and len(stdout_tx0_ss) and len(stdout_tx1_json) and len(stdout_tx1_ss):
        print("Already have " + name_tx0 + " and "+ name_tx1 + ", skipping")

    elif len(stderr_tx0_json) or len(stderr_tx0_ss) or len(stderr_tx1_json) or len(stderr_tx1_ss):
        print("Running experiment to generate " + name_tx0 + " and "+ name_tx1)
```

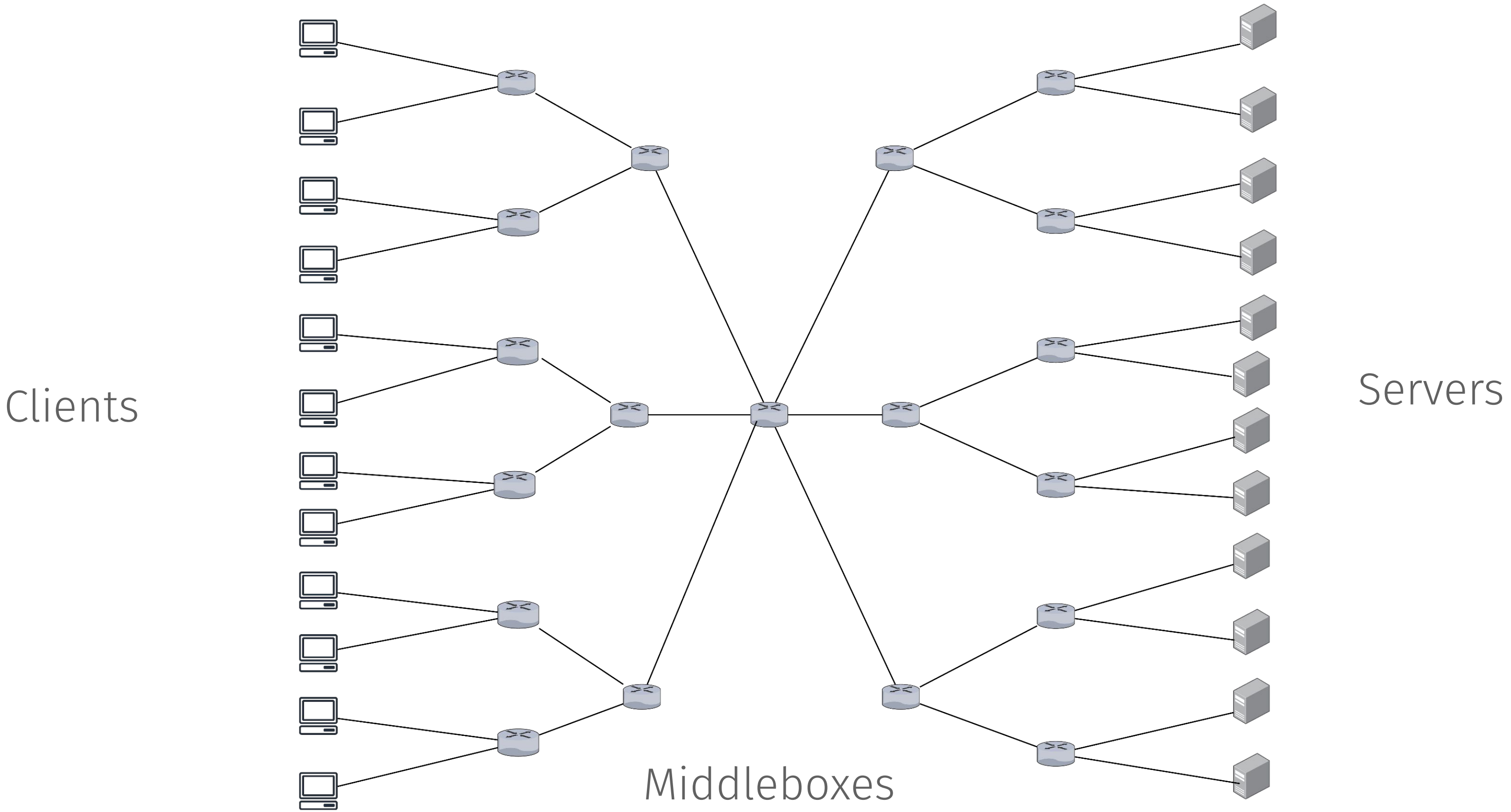
Process data, retrieve to Jupyter env for visualization

```
plt.figure(fig1.number)
fig1.suptitle('100 Mbps - 20ms RTT- Server A ECN: AccECN - Server B ECN: Classic ECN - ECN Fallback: ON', fontsize=45)
#fig1.subplots_adjust(hspace=0.15)  # Adjust the vertical spacing
cbar_ax = fig1.add_axes([0.92, 0.15, 0.02, 0.7])  # x-position, y-position, width, height
cbar=fig1.colorbar(ax1.collections[0], cax=cbar_ax)
cbar.set_label('Prague Throughput Share', size=40)
cbar.ax.tick_params(labelsize=35)
plt.tight_layout(rect=[0, 0.01, 0.9, 0.98], pad=1.08, h_pad=1.5, w_pad=1.08)
plt.savefig('/home/fabric/work/throughput.png', dpi=100)
```

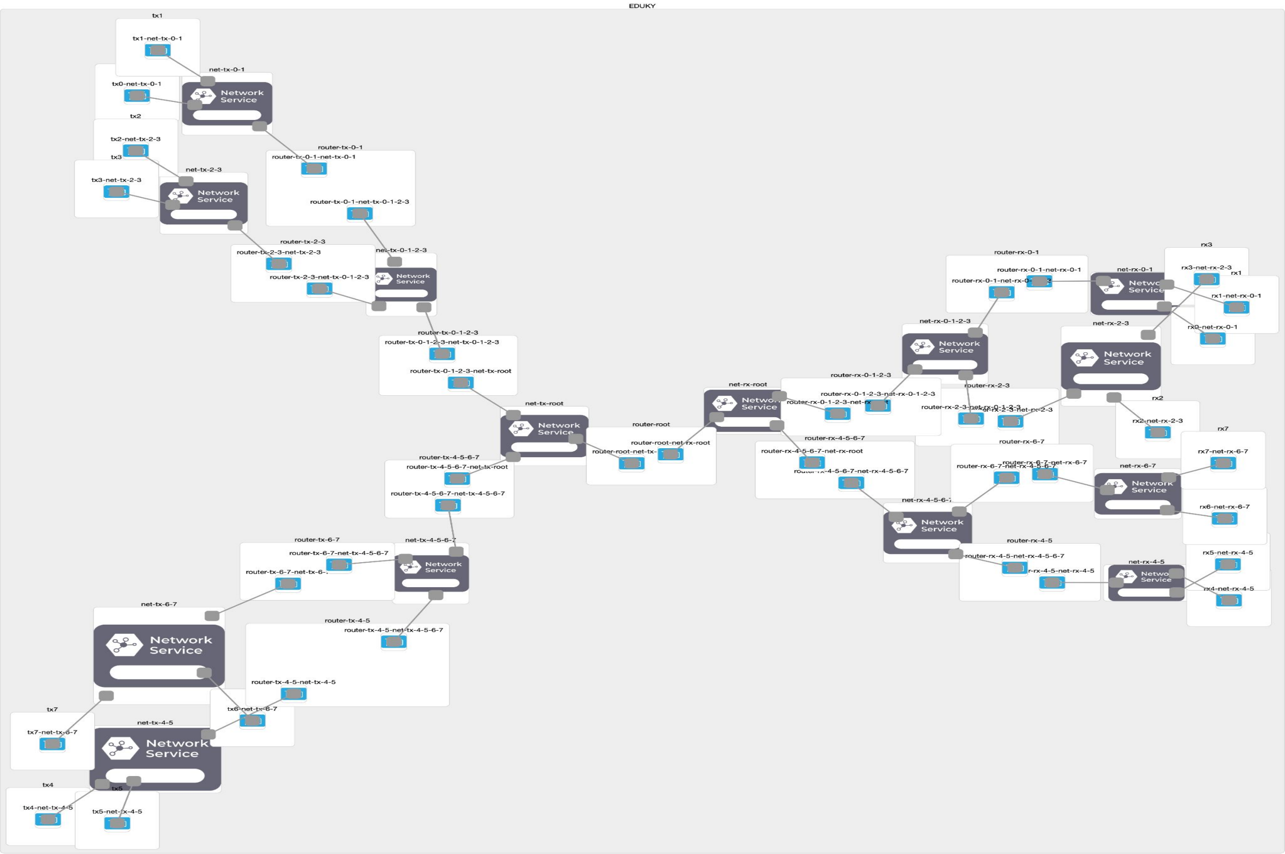
```
/tmp/ipykernel_152/3515026403.py:218: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout(rect=[0, 0.01, 0.9, 0.98], pad=1.08, h_pad=1.5, w_pad=1.08)
```



Network with multiple bottlenecks



Network with multiple bottlenecks



Configuring a large topology

```python
# Initialize the node configuration list
node_conf = []

# Generate tx and rx nodes
for i in range(N):  # tx0...tx11 and rx0...rx11
    tx_node = base_config.copy()
    tx_node['name'] = f"tx{i}"
    node_conf.append(tx_node)

    rx_node = base_config.copy()
    rx_node['name'] = f"rx{i}"
    node_conf.append(rx_node)

# Generate router-tx and router-rx nodes
for i in range(0, N, 2):  # pairs (0-1, 2-3, ..., 10-11)
    router_tx_node = base_config.copy()
    router_rx_node = base_config.copy()
    router_tx_node['name'] = f"router-tx-{i}-{i+1}"
    router_rx_node['name'] = f"router-rx-{i}-{i+1}"
    node_conf.append(router_tx_node)
    node_conf.append(router_rx_node)

# Generate router-tx and router-rx nodes for groups of four
for i in range(0, N, 4):  # groups of four (0-1-2-3, 4-5-6-7, ...)
    router_tx_node = base_config.copy()
    router_rx_node = base_config.copy()
    router_tx_node['name'] = f"router-tx-{i}-{i+1}-{i+2}-{i+3}"
    router_rx_node['name'] = f"router-rx-{i}-{i+1}-{i+2}-{i+3}"
    node_conf.append(router_tx_node)
    node_conf.append(router_rx_node)

# Add the router-root node
router_root_node = base_config.copy()
router_root_node['name'] = "router-root"
node_conf.append(router_root_node)
```

Configure node, network and routing configurations in a loop!

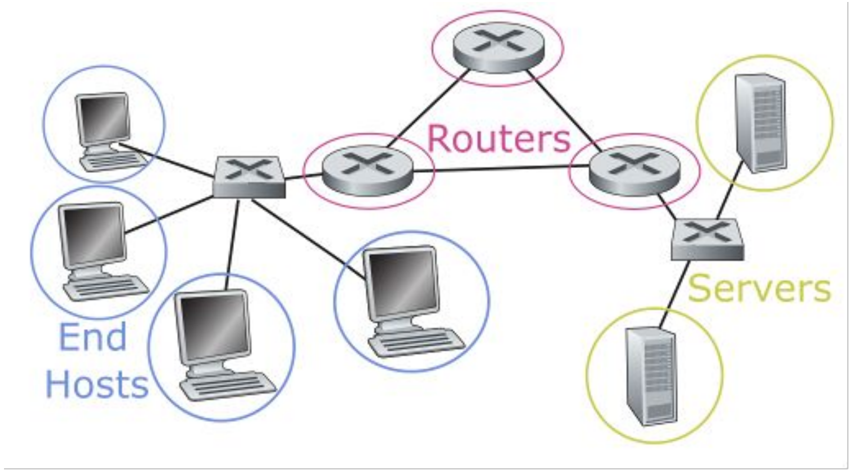
Outline

- Introduction
  - Background for L4S
  - L4S deployment requirements
  - Problem Statement
- How does FABRIC support this research?
  - Single bottleneck experiments
  - Multiple bottleneck experiments
- Initial Findings / Results
- Conclusion

Research question

What would the Internet look like with a partial L4S deployment?

Would a partial deployment encourage further deployment, or encourage reversion to status quo?



Experimental Analysis

- Evaluation of the coexistence of TCP Prague with TCP Cubic under various network conditions
- Fairness & Queuing Delay analysis
- Checking the validity of the points discussed about L4S coexistence

Performance Evaluation Metrics

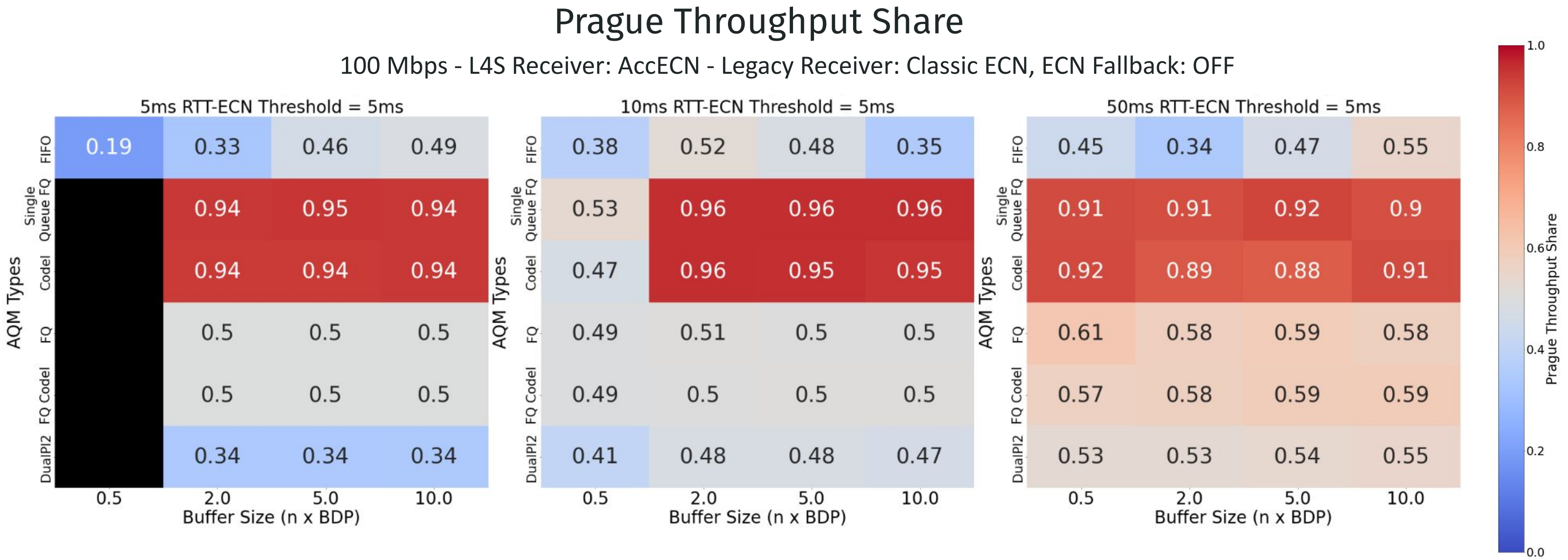
Prague Throughput Share: (a value greater than 0.5 means Prague gets more throughput than CUBIC)

$$\text{PragueShare} = \frac{\text{Throughput}|_{\text{Prague}}}{\text{Throughput}|_{\text{Prague}} + \text{Throughput}|_{\text{Cubic}}}$$

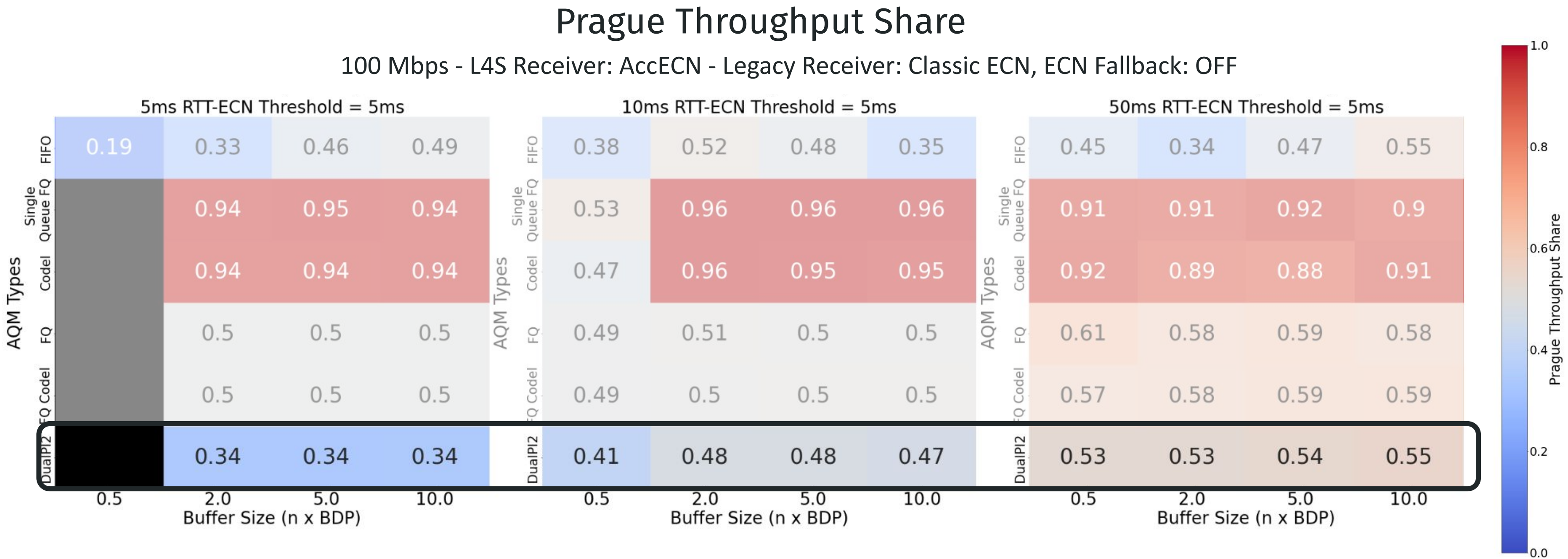
Cubic Relative Queueing Delay: (a value closer to 1 indicates higher queuing delay for CUBIC)

$$\text{QueueingDelayGain}_{\text{Cubic}}^{\text{Prague}} = \frac{\text{QueueingDelay}|_{\text{Cubic}} - \text{QueueingDelay}|_{\text{Prague}}}{\text{QueueingDelay}|_{\text{Cubic}}}$$

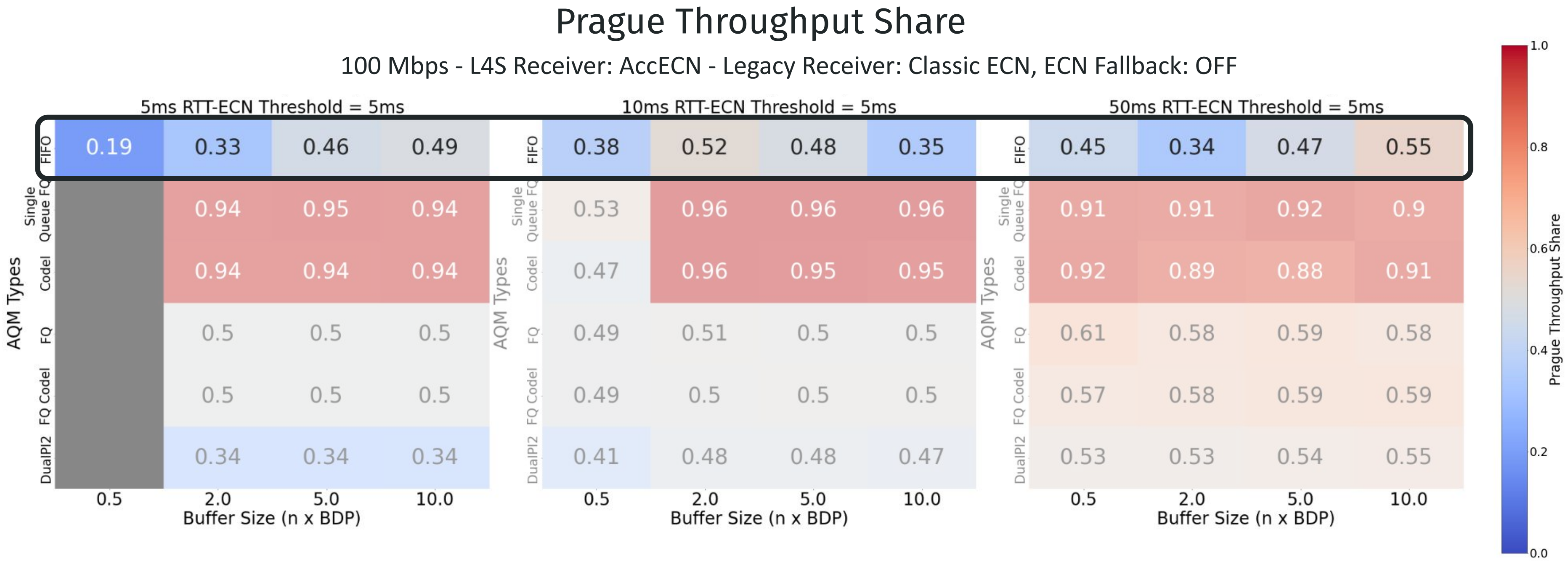
Experiment Results



When L4S is fully deployed, the range of outcomes goes from ‘legacy does slightly better’ to ‘L4S does slightly better’

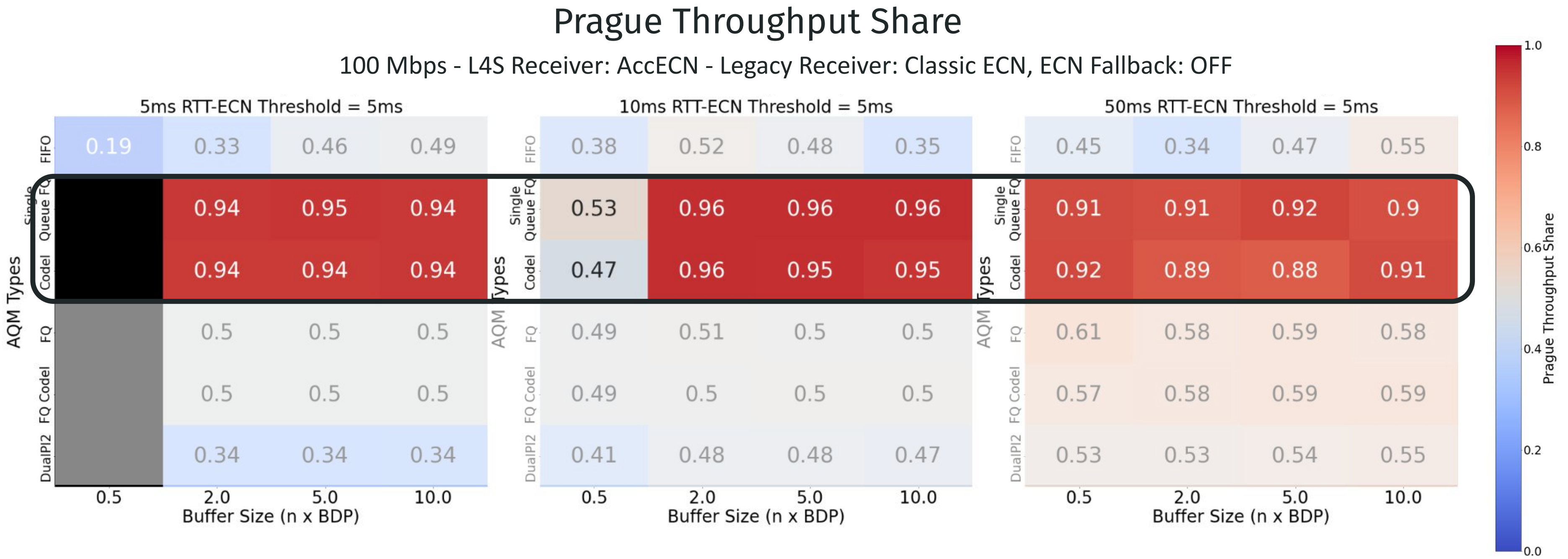


In a FIFO bottleneck without ECN, legacy flows do about as well or a bit better than L4S flows.

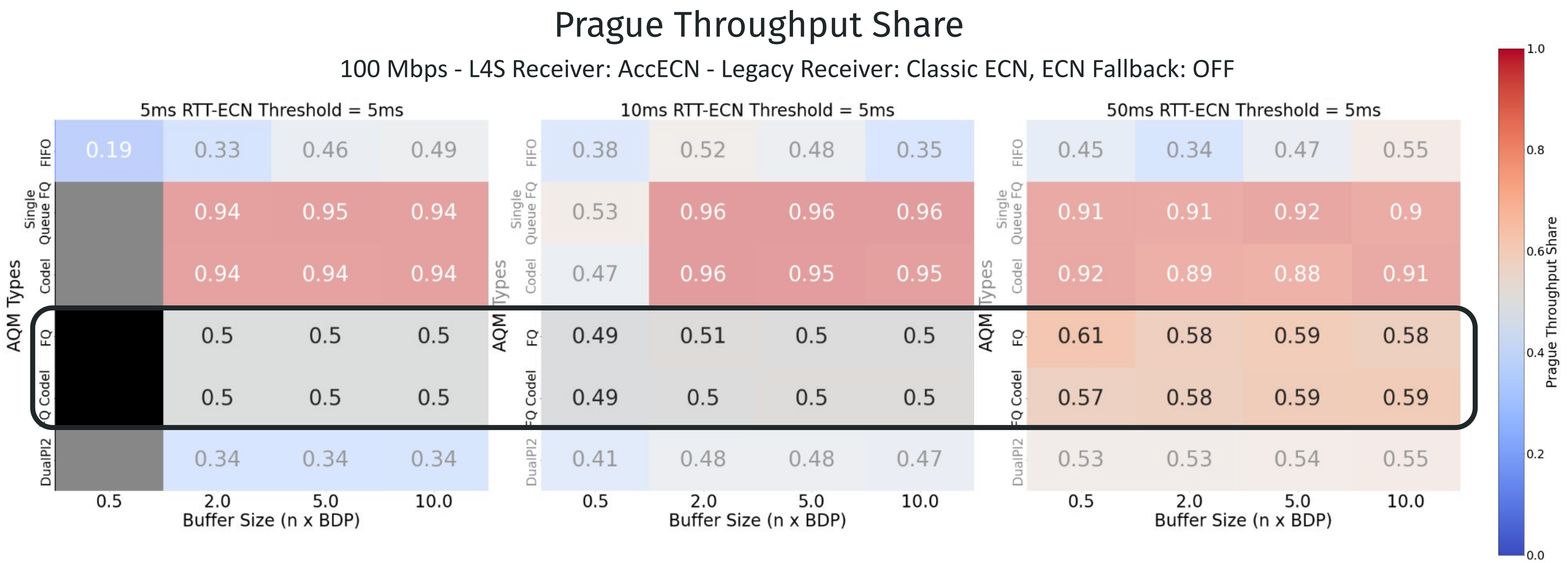


AQM Type	5ms: 0.5	5ms: 2.0	5ms: 5.0	5ms: 10.0	10ms: 0.5	10ms: 2.0	10ms: 5.0	10ms: 10.0	50ms: 0.5	50ms: 2.0	50ms: 5.0	50ms: 10.0
FIFO	0.19	0.33	0.46	0.49	0.38	0.52	0.48	0.35	0.45	0.34	0.47	0.55
Single Queue FQ		0.94	0.95	0.94	0.53	0.96	0.96	0.96	0.91	0.91	0.92	0.9
Codel		0.94	0.94	0.94	0.47	0.96	0.95	0.95	0.92	0.89	0.88	0.91
FQ		0.5	0.5	0.5	0.49	0.51	0.5	0.5	0.61	0.58	0.59	0.58
FQ Codel		0.5	0.5	0.5	0.49	0.5	0.5	0.5	0.57	0.58	0.59	0.59
DualPI2		0.34	0.34	0.34	0.41	0.48	0.48	0.47	0.53	0.53	0.54	0.55

When bottleneck has ECN but no isolation for flow types, L4S will dominate (and legacy flows suffer from underutilization)

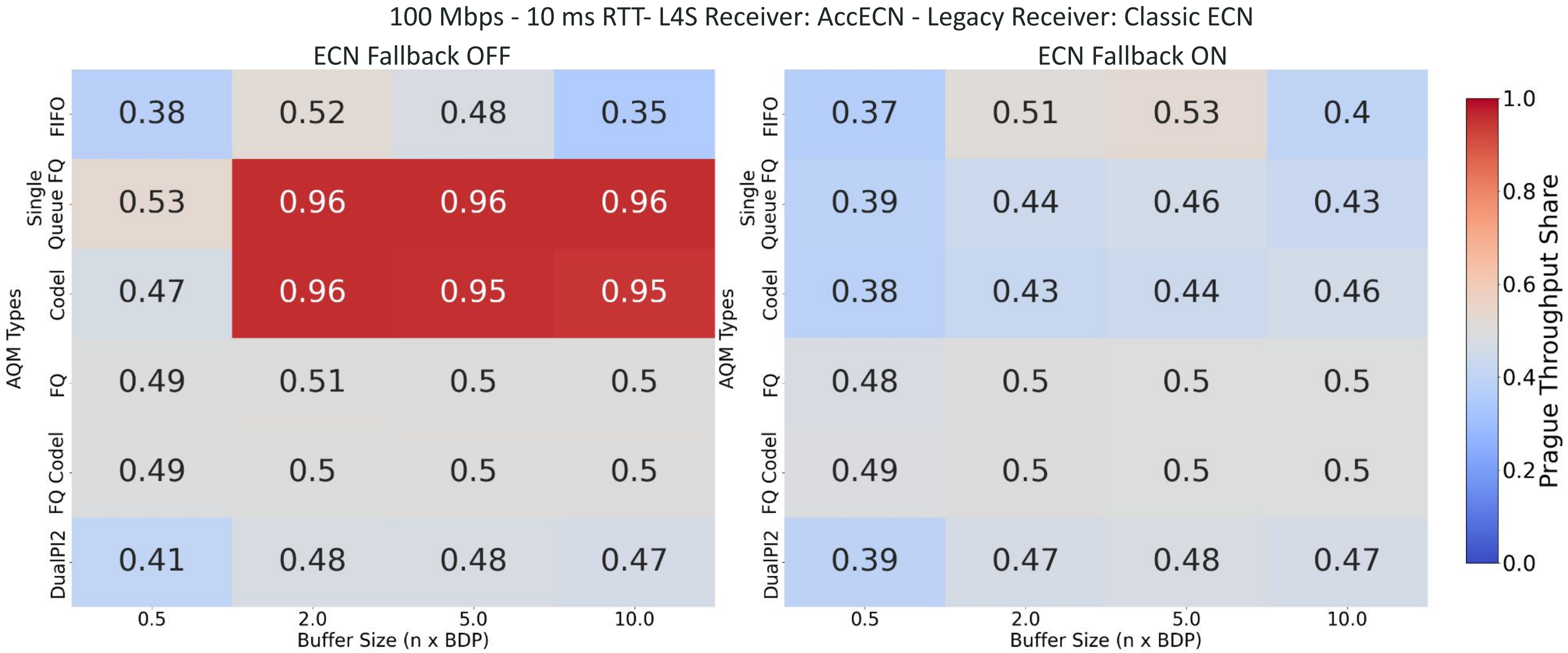


A fair queue or dual queue AQM ensures that legacy and L4S flows get approximately equal share.



ECN Fallback Algorithm Results

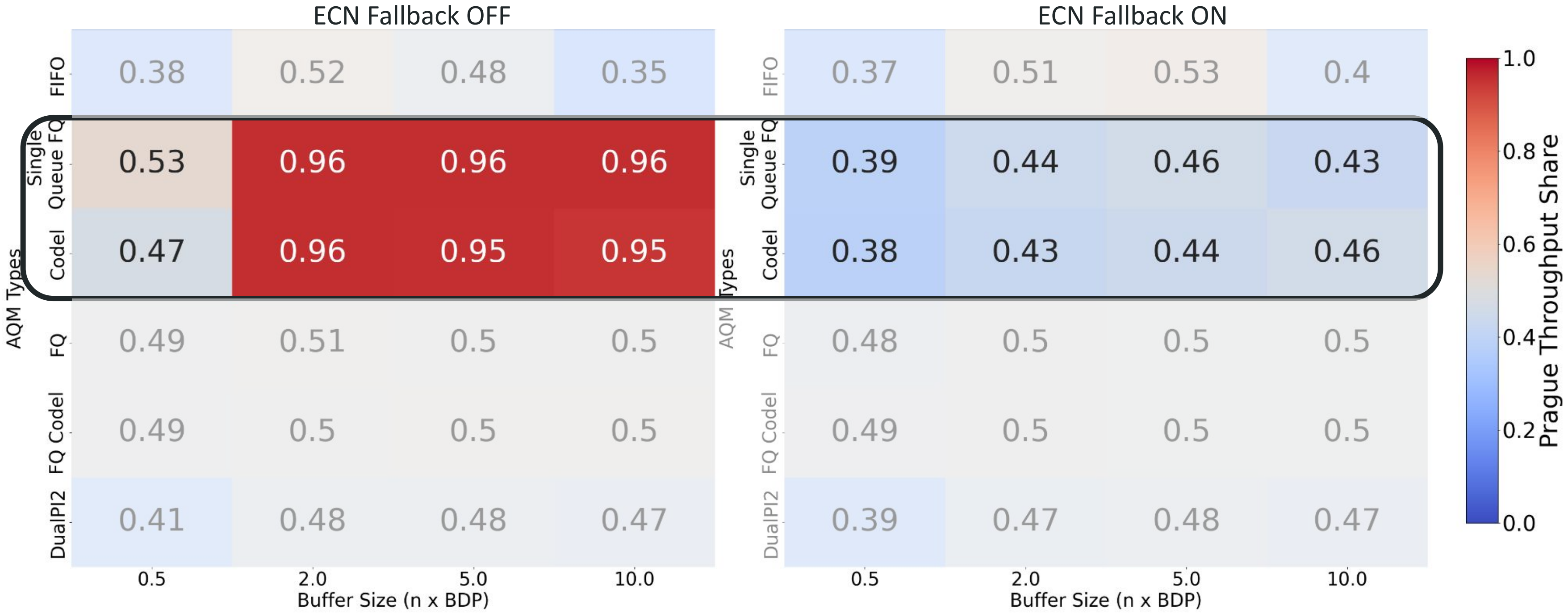
ECN Fallback Algorithm



ECN Fallback Algorithm addresses the domination of L4S flows problem in the bottlenecks with ECN and no flow isolation

ECN Fallback Algorithm

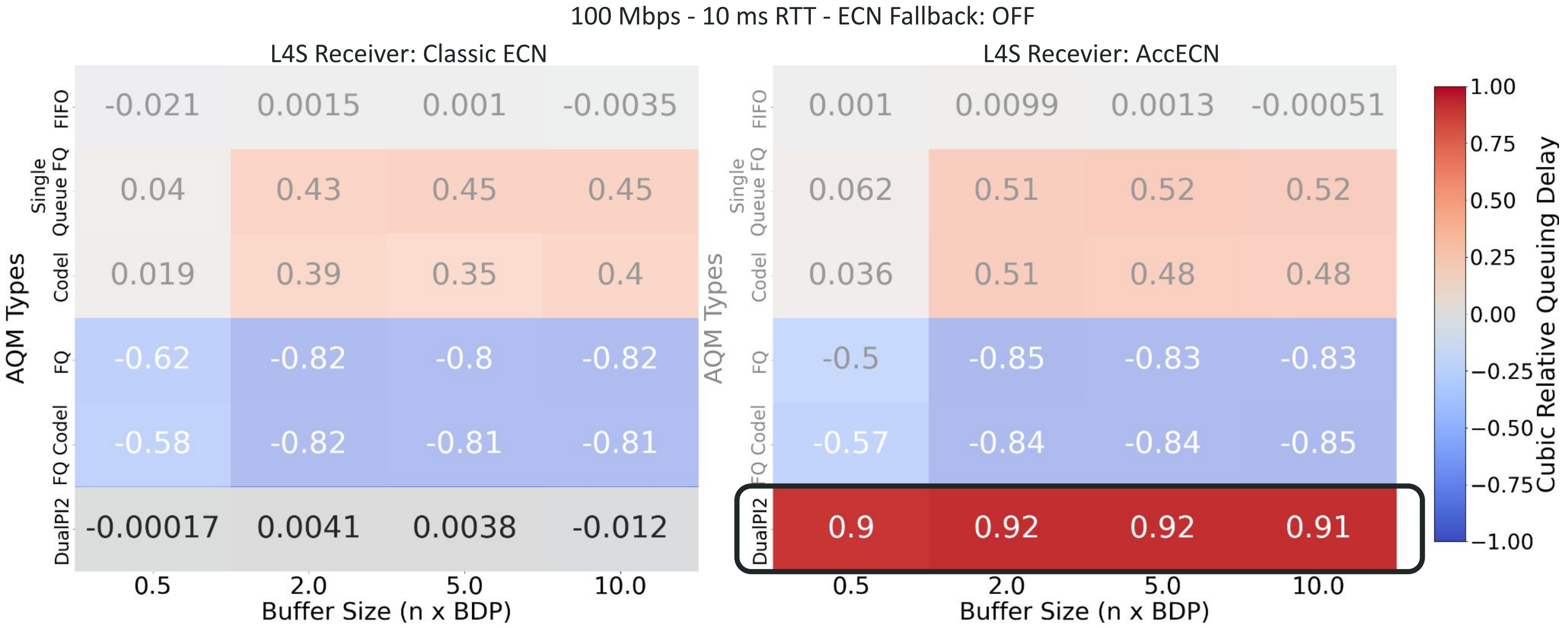
100 Mbps - 10 ms RTT- L4S Receiver: AccECN - Legacy Receiver: Classic ECN



But, there are other issues when the ECN threshold is low!

No latency benefit unless there is full L4S deployment

CUBIC Relative Queuing Delay



Outline

- Introduction
  - Background for L4S
  - L4S deployment requirements
  - Problem Statement
- How does FABRIC support this research?
  - Single bottleneck experiments
  - Multiple bottleneck experiments
- Initial Findings / Results
- Conclusion

Conclusion

- In some scenarios, legacy flows are better off.
- L4S flows harm legacy flows in many scenarios.
- Latency benefit is only obtained when there is full L4S support (Scalable Congestion Control, Dual Queue AQM, AccECN)

Thank You for Listening !

Contact: fbs6417@nyu.edu

References

- [1] O. Albisser, K. De Schepper, B. Briscoe, O. Tilmans, and H. Steen, “DUALPI2—Low Latency, Low Loss and Scalable (L4S) AQM,” Net-Dev 0x13, Prague, 2019.
- [2] B. Briscoe, K. De Schepper, M. Bagnulo, and G. White, “RFC 9330: Low latency, low loss, and scalable throughput (L4S) internet service: Architecture,” USA, 2023.
- [3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” SIGCOMM Comput. Commun. Rev., vol. 40, no. 4, p. 63–74, aug 2010. [Online]. Available: https://doi.org/10.1145/1851275.1851192
- [4] B. Briscoe, K. D. Schepper, O. Albisser, O. Tilmans, N. Kuhn, G. Fairhurst, R. Scheffenegger, M. Abrahamsson, I. Johansson, P. Balasubramanian, D. Pullen, G. Bracha, J. Morton, and D. Ta¨ht, “Implementing the ’ prague requirements ’ for low latency low loss scalable throughput (l 4 s),” 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:180259180
- [5] K. D. Schepper, O. Albisser, O. Tilmans, and B. Briscoe, “Dual queue coupled aqm: Deployable very low queuing delay for all,” 2022.
- [6] B. Briscoe, M. Ku¨hlewind, and R. Scheffenegger, “More Accurate Explicit Congestion Notification (ECN) Feedback in TCP,” Internet Engineering Task Force, Internet-Draft draft-ietf-tcpm- accurate-ecn-28, Nov. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tcpm-accurate-ecn/28/
- [7] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, “The Addition of Explicit Congestion Notification (ECN) to IP,” RFC 3168, Sep. 2001. [Online]. Available: https://www.rfc-editor.org/info/rfc3168

References

- [8] K. D. Schepper, B. Briscoe, and G. White, “Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S),” RFC 9332, Jan. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9332
- [9] K. D. Schepper and B. Briscoe, “The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S),” RFC 9331, Jan. 2023. [Online]. Available: https://www.rfc- editor.org/info/rfc9331
- [10] B. Briscoe and A. S. Ahmed, “Tcp prague fall-back on detection of a classic ecn aqm,” 2021.
- [11] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, “Fabric: A national-scale programmable experimental network infrastructure,” IEEE Internet Computing, vol. 23, no. 6, pp. 38–47, 2019.

Why FABRIC?

- This project is of interest to the FABRIC community as an example of a large-scale, systematic, and reproducible experiment enabled by the FABRIC infrastructure.

- Our work involves a large number of complicated experiments, and the Jupyter notebook interface and Python library provided by FABRIC are very useful for handling them.

- This presentation includes a section titled 'How does FABRIC support this research?' which provides some useful tricks for handling large and complicated experiments.

FABRIC