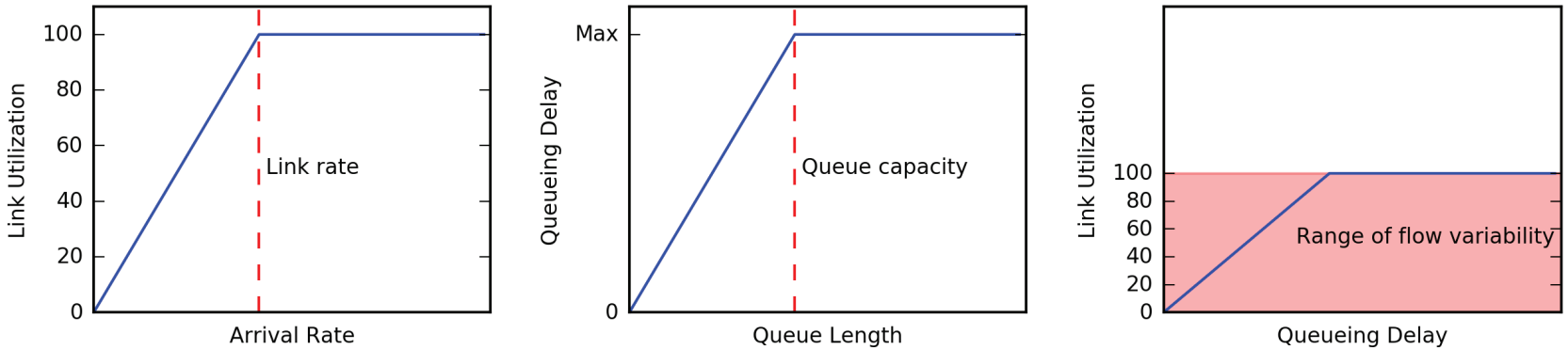
Assumed Relationship Between Utilization and Delay



- Linear relation between queue length and queueing delay
- Queueing delay bounds are enough to cover flow variability

Transport Layer Congestion Control

TCP at Hosts + Queue Management at Bottleneck Link

TCP

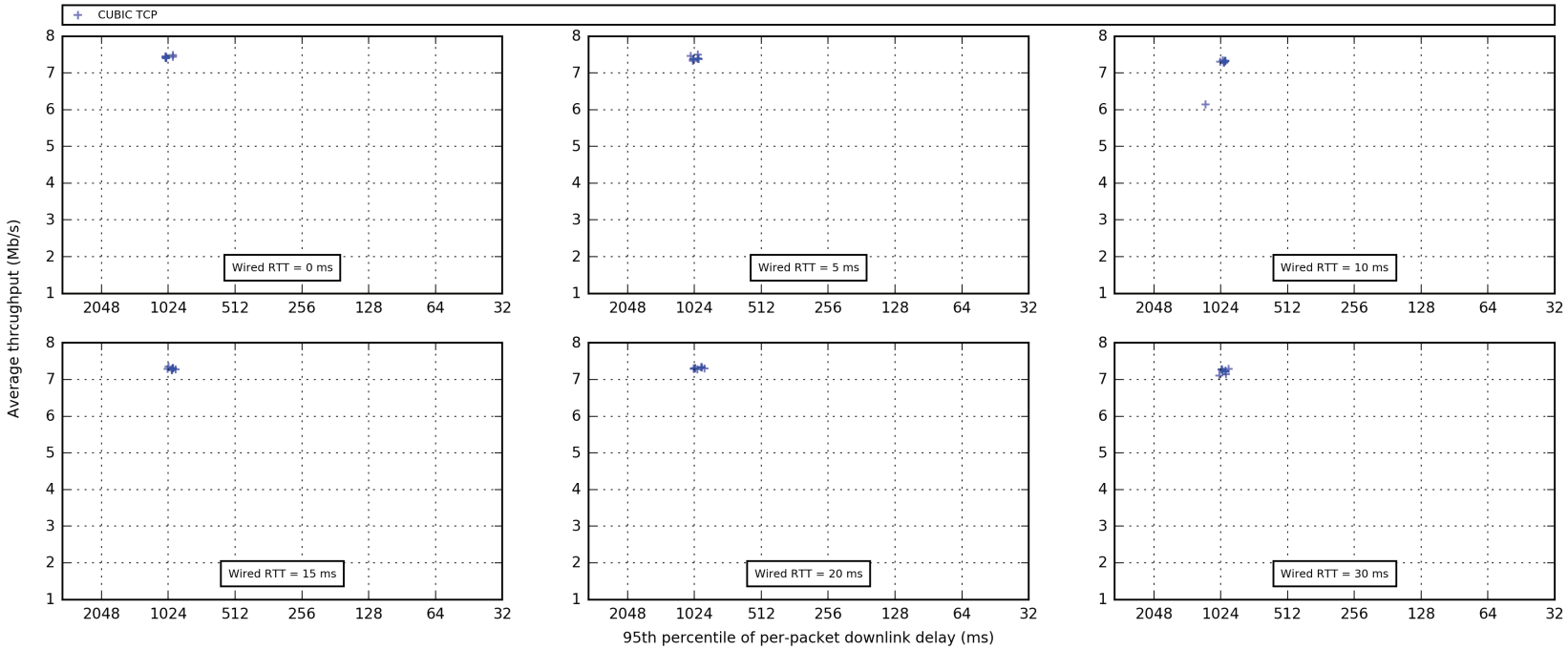
- Loss-based
  → maintain full queue
- Delay-based
  → maintain shallow queue
- Hybrid
  → match estimated bottleneck capacity

Bottleneck Queue

- Droptail
  → mantain full utilization
- CoDel
  → maintain delay bound
- PIE
  → maintain delay bound

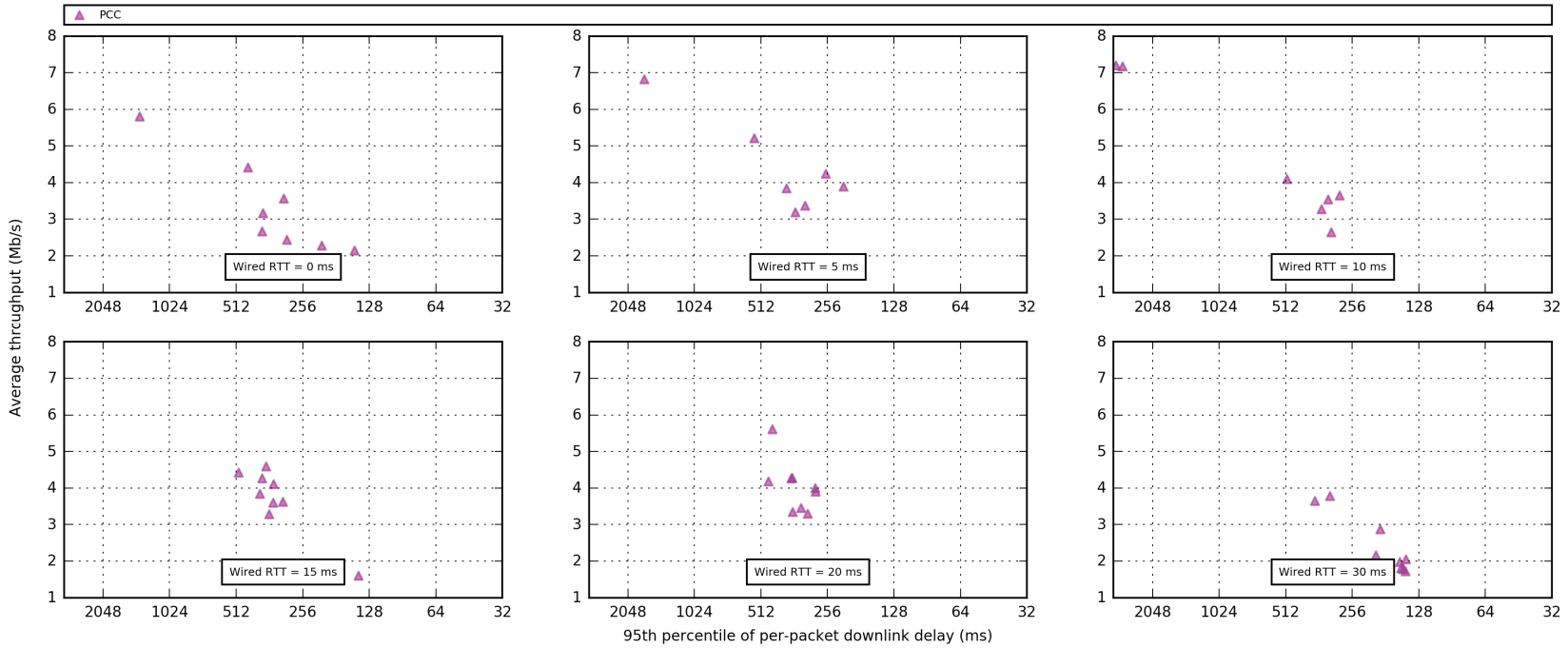
Outline

Performance of Existing Solutions



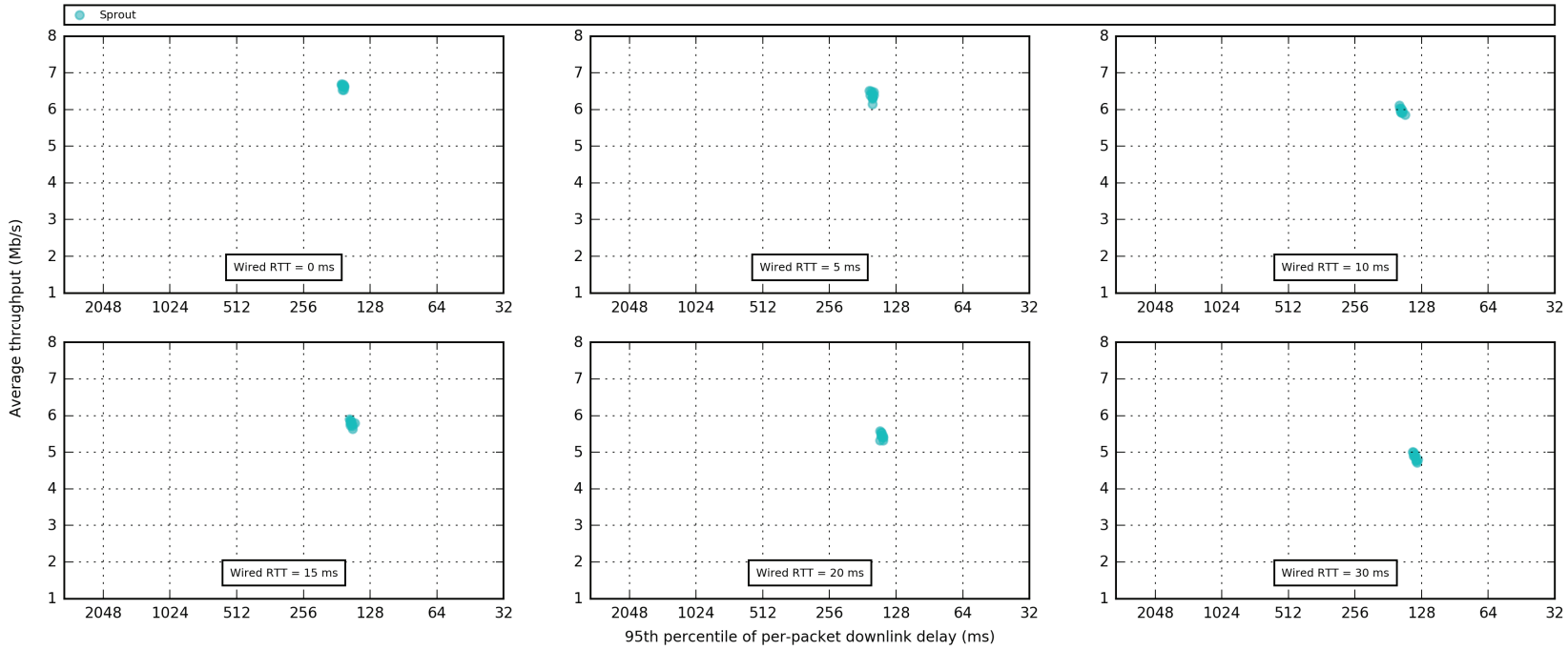
- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

Performance of Existing Solutions



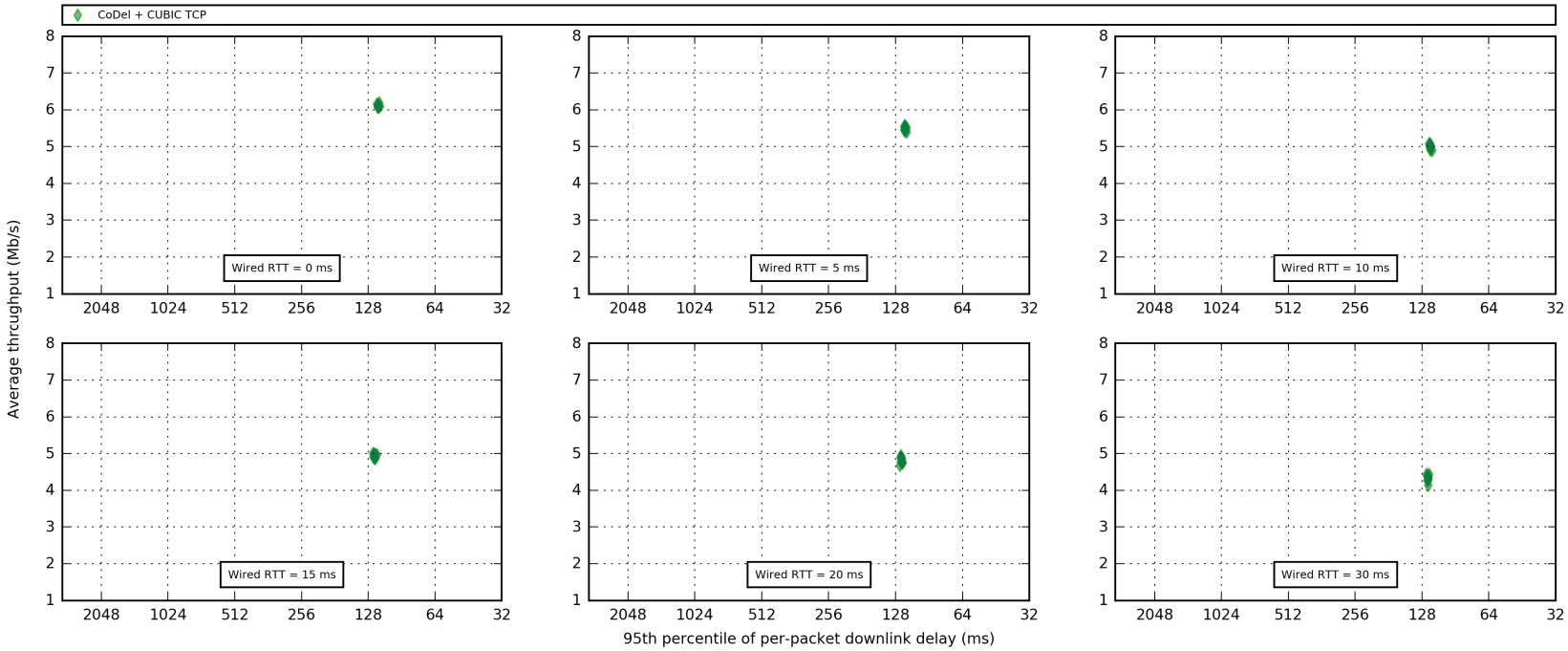
- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

Performance of Existing Solutions



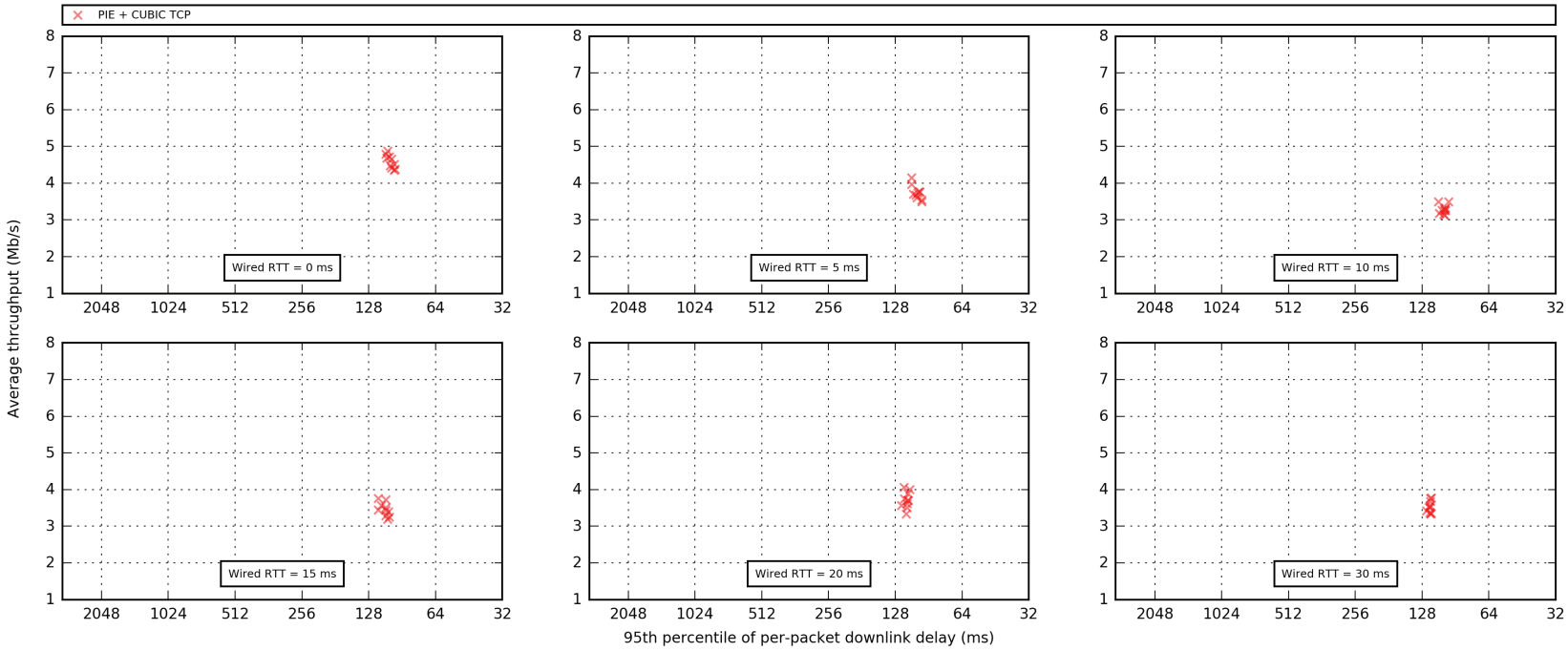
- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

Performance of Existing Solutions



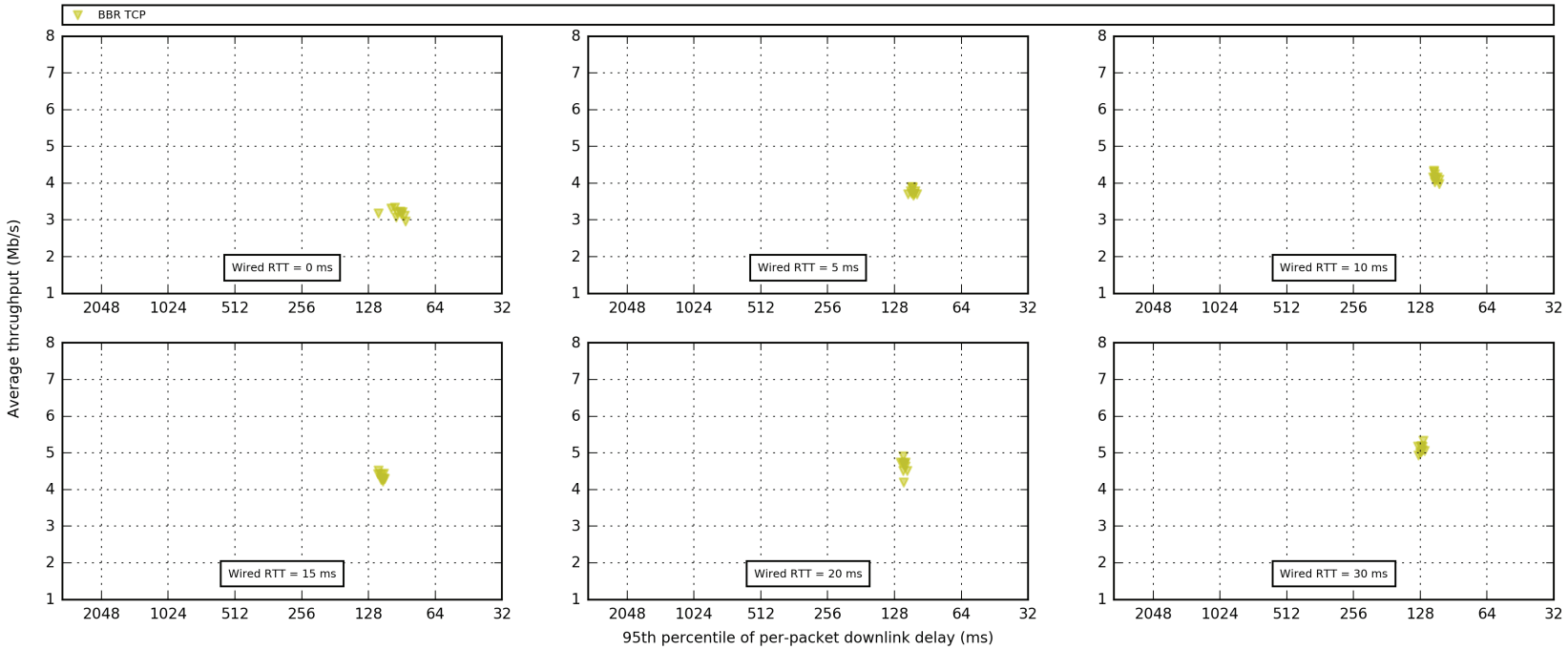
- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

Performance of Existing Solutions



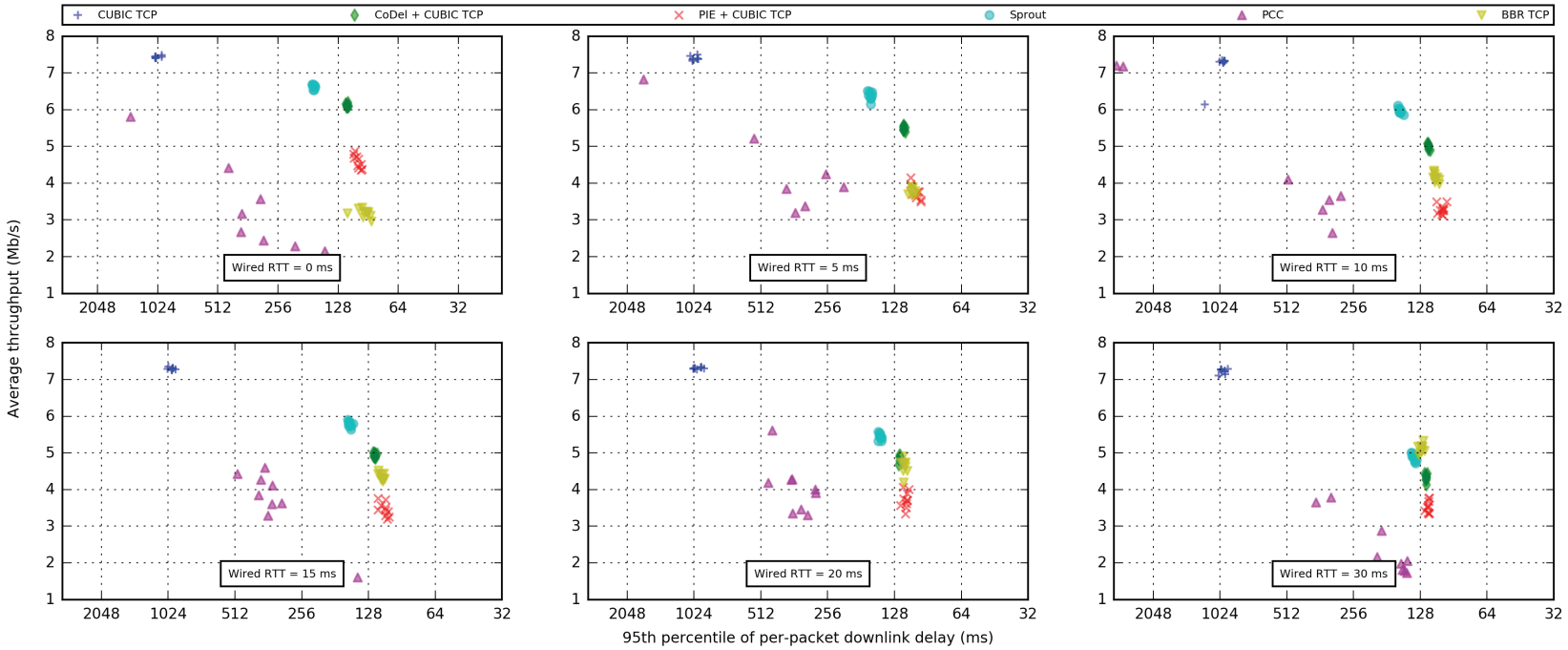
- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

Performance of Existing Solutions



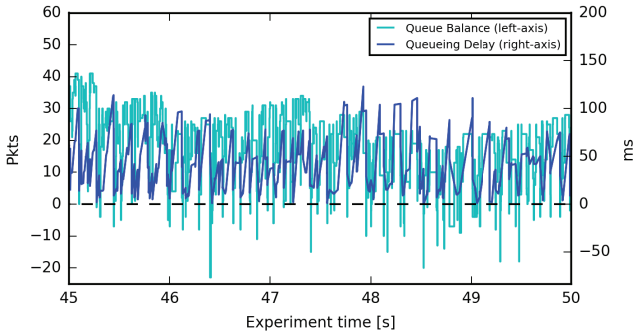
- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

Performance of Existing Solutions

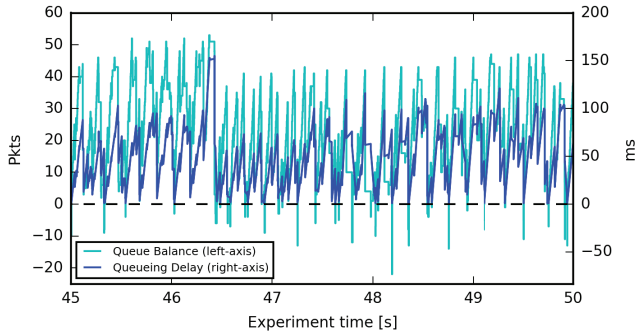


- Effect of changing RTT
- Inconsistent results
- Utilization vs. delay trade-off
- Distance from optimum

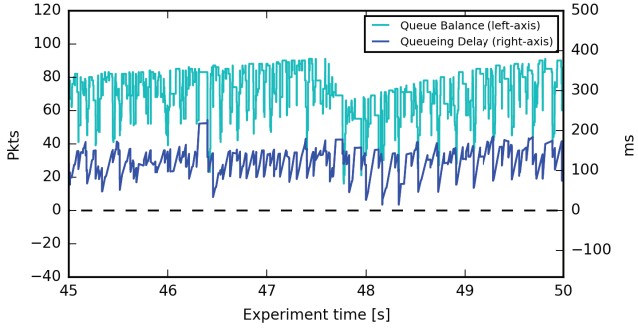
Queue Balance



CoDel

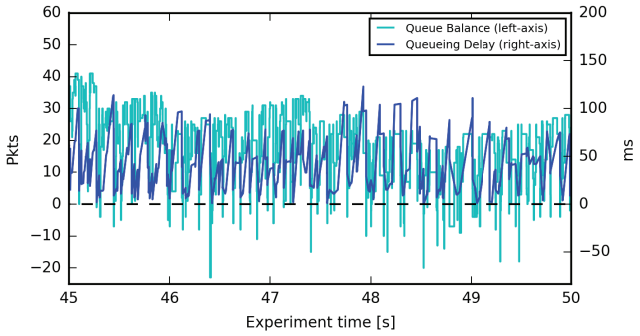


BBR TCP

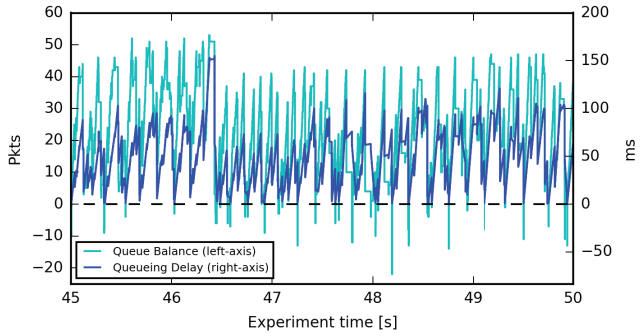


80-packet droptail

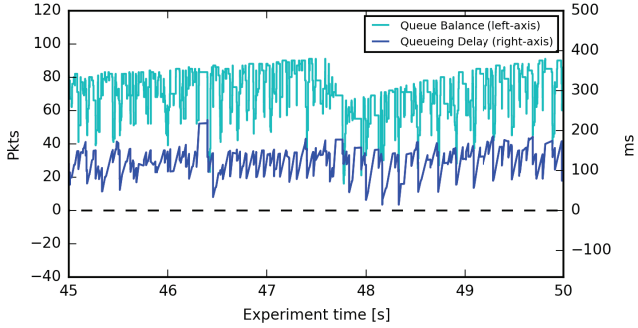
Queue Balance



CoDel



BBR TCP



80-packet droptail

- Existing methods cannot estimate queue balance accurately
- Limited accuracy of network congestion signals
- Lack of new network capacity signal

Goal

What we have

- Throughput-oriented solutions
- Delay-oriented solutions
- Complicated flow coexistence interactions
- Conflicting goals for network and application providers

What we want

- Individual application utilization vs. delay tradeoff based on queue balance
- Flow isolation to avoid unintended interactions
- Joint optimization of network and application provider goals

Goal

What we have

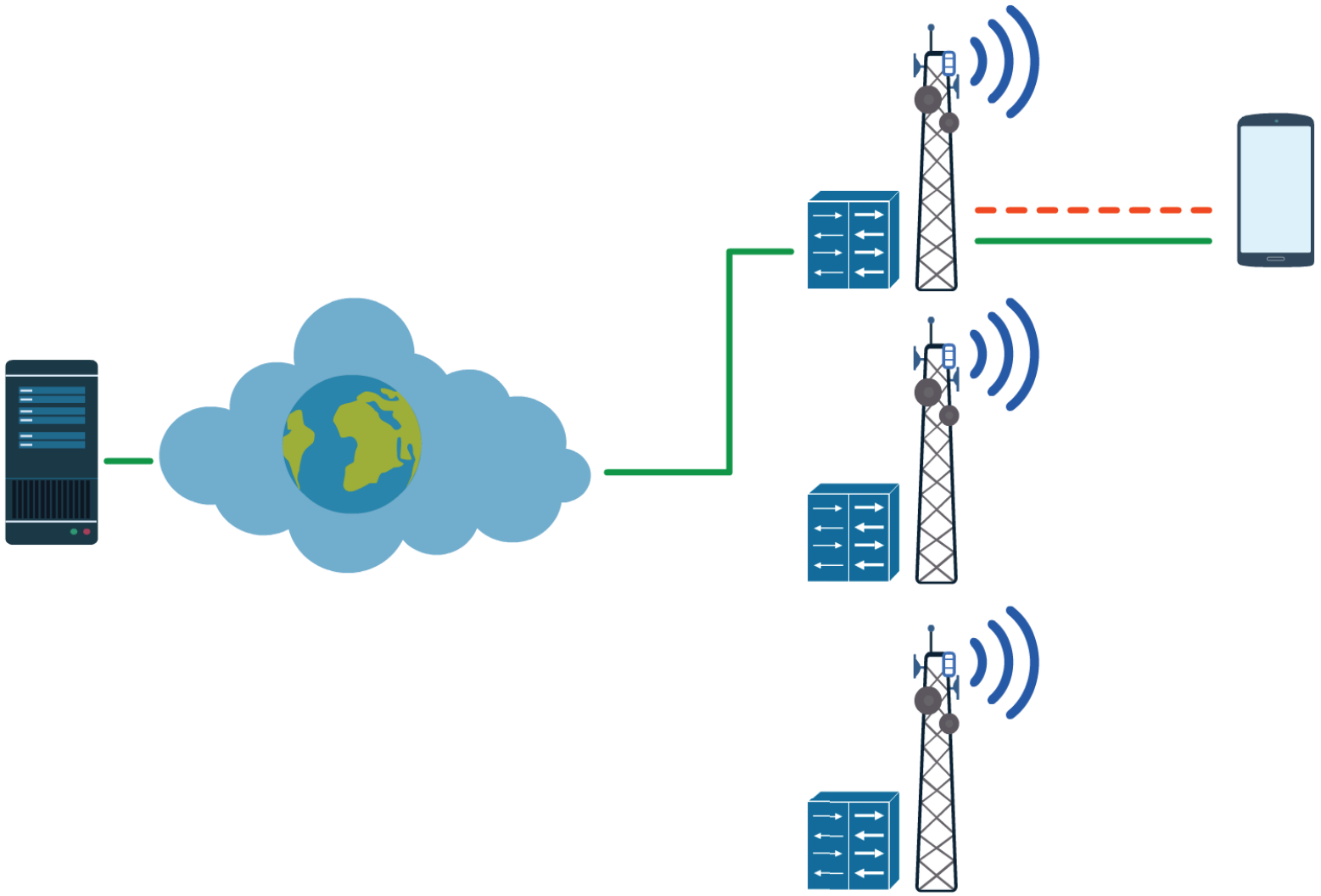
- Throughput-oriented solutions
- Delay-oriented solutions
- Complicated flow coexistence interactions
- Conflicting goals for network and application providers

What we want

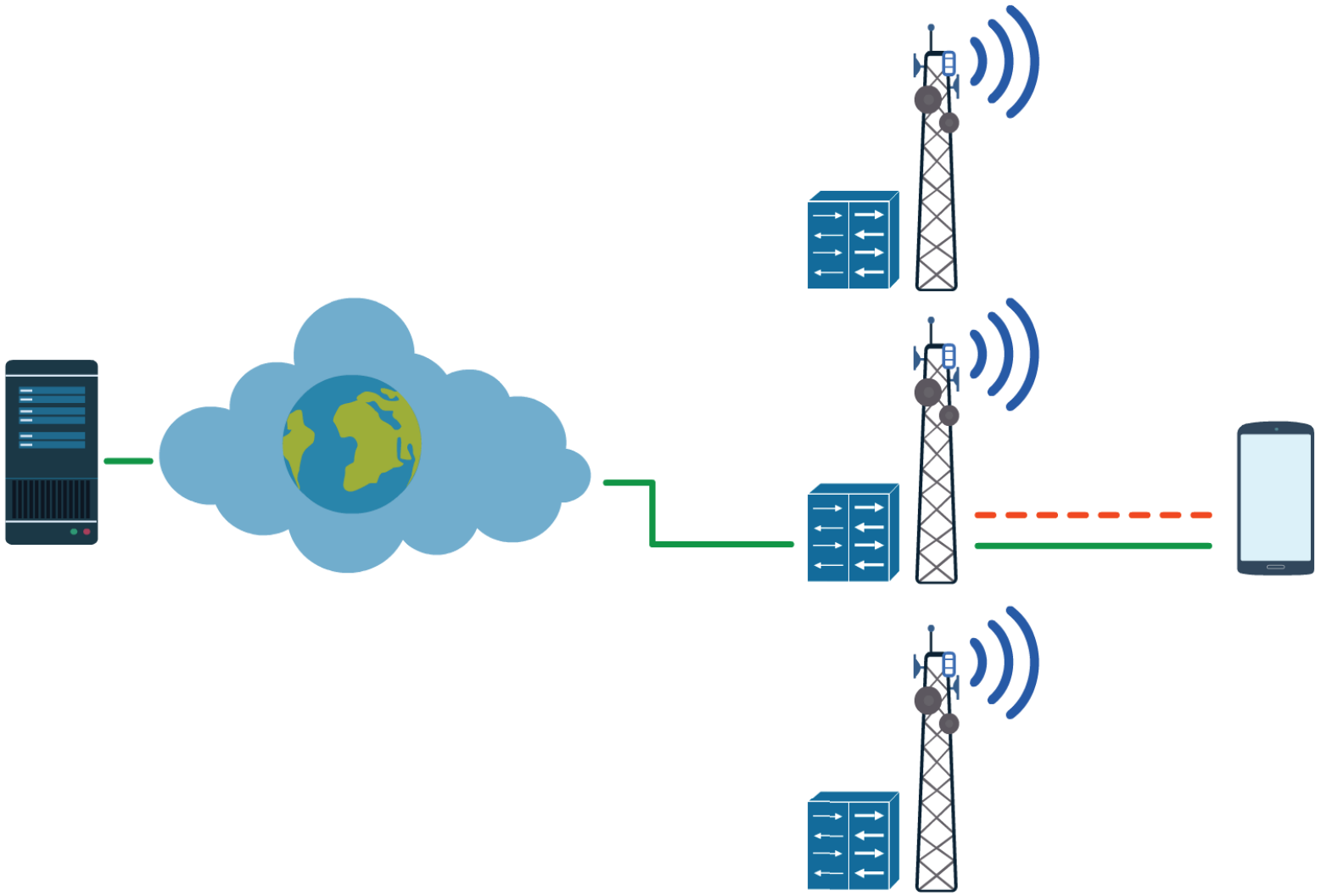
- Individual application utilization vs. delay tradeoff based on queue balance
- Flow isolation to avoid unintended interactions
- Joint optimization of network and application provider goals

Opportunity! Emerging trends in upcoming 5G networks

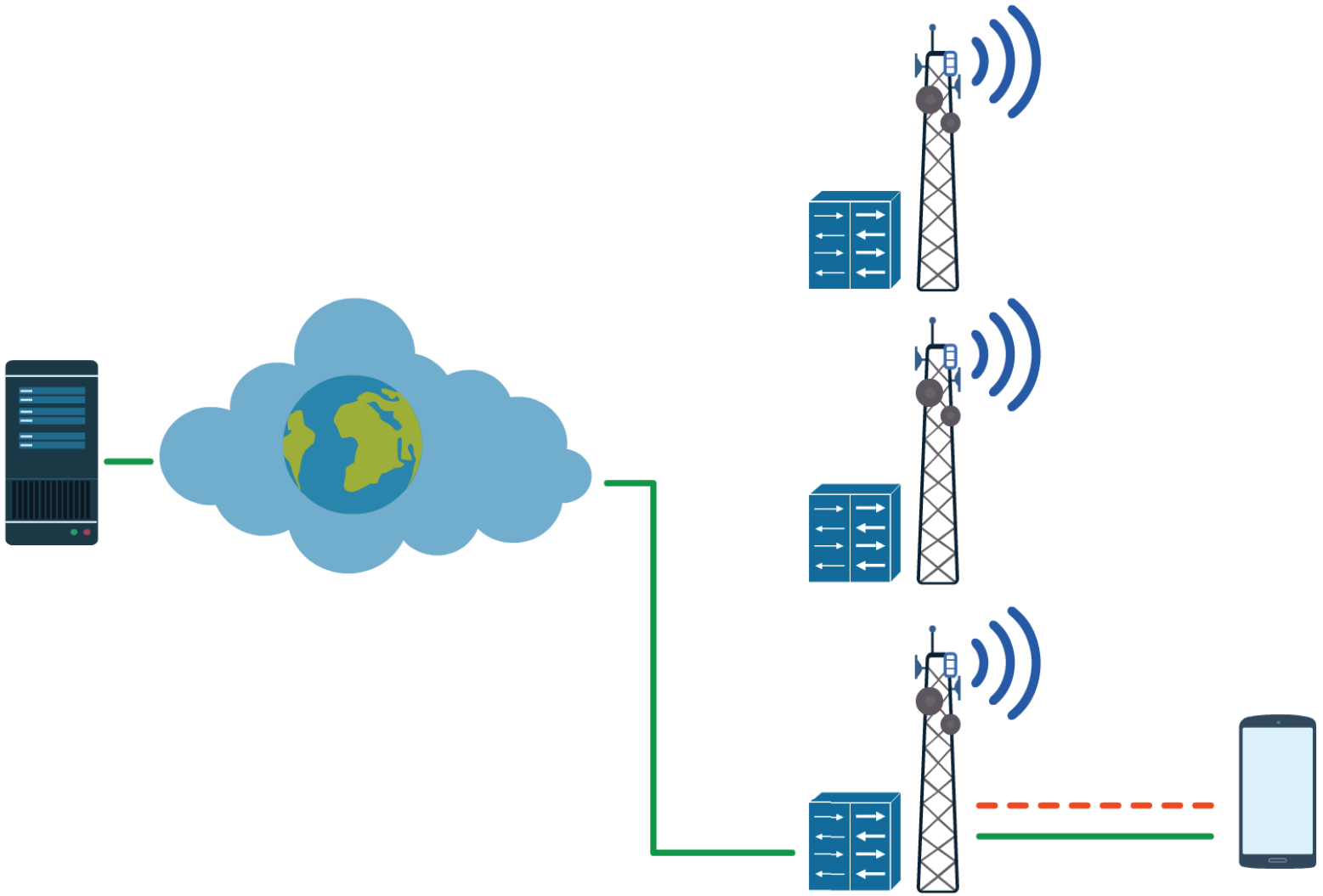
Emerging trend in 5G wireless access



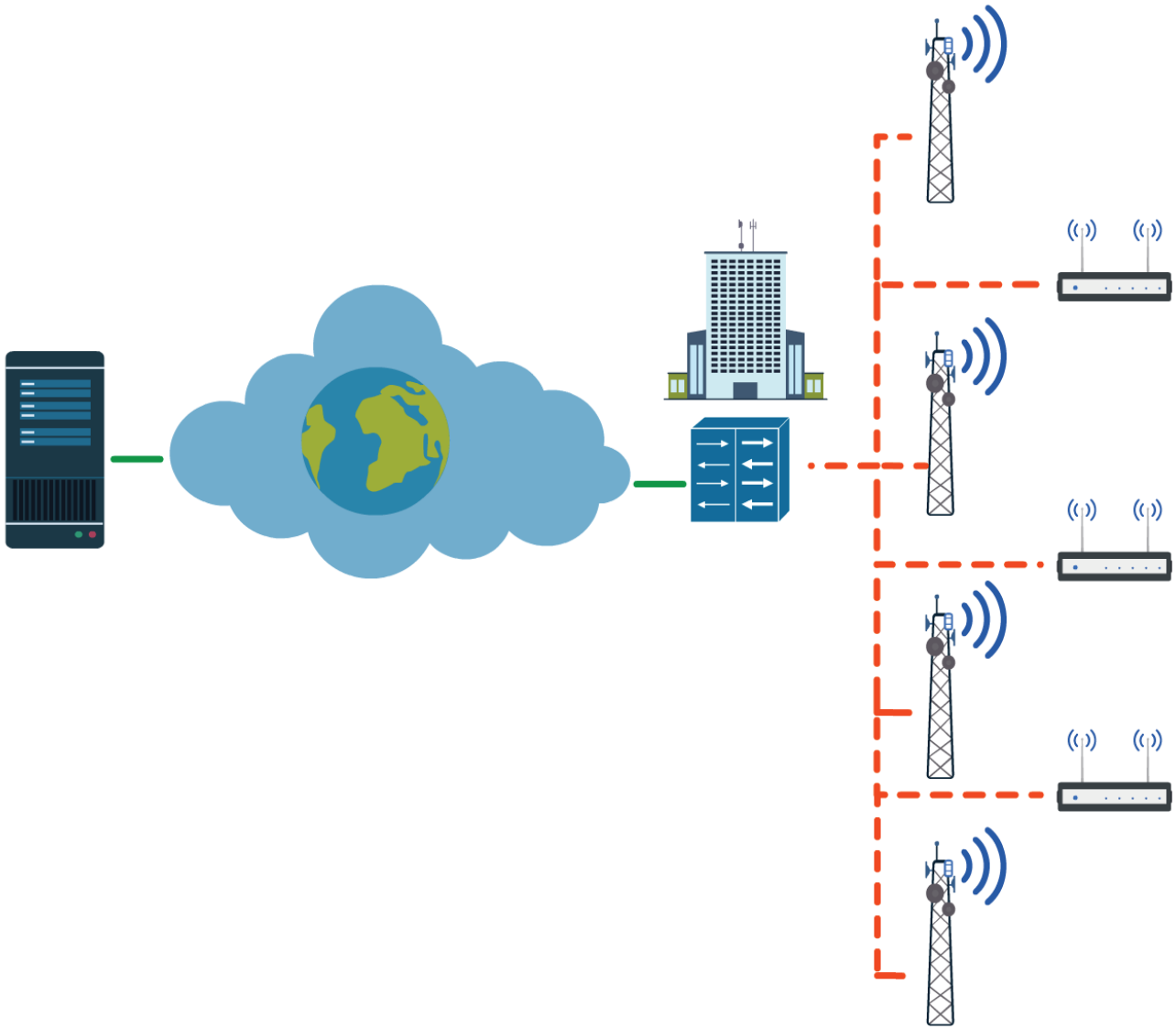
Emerging trend in 5G wireless access



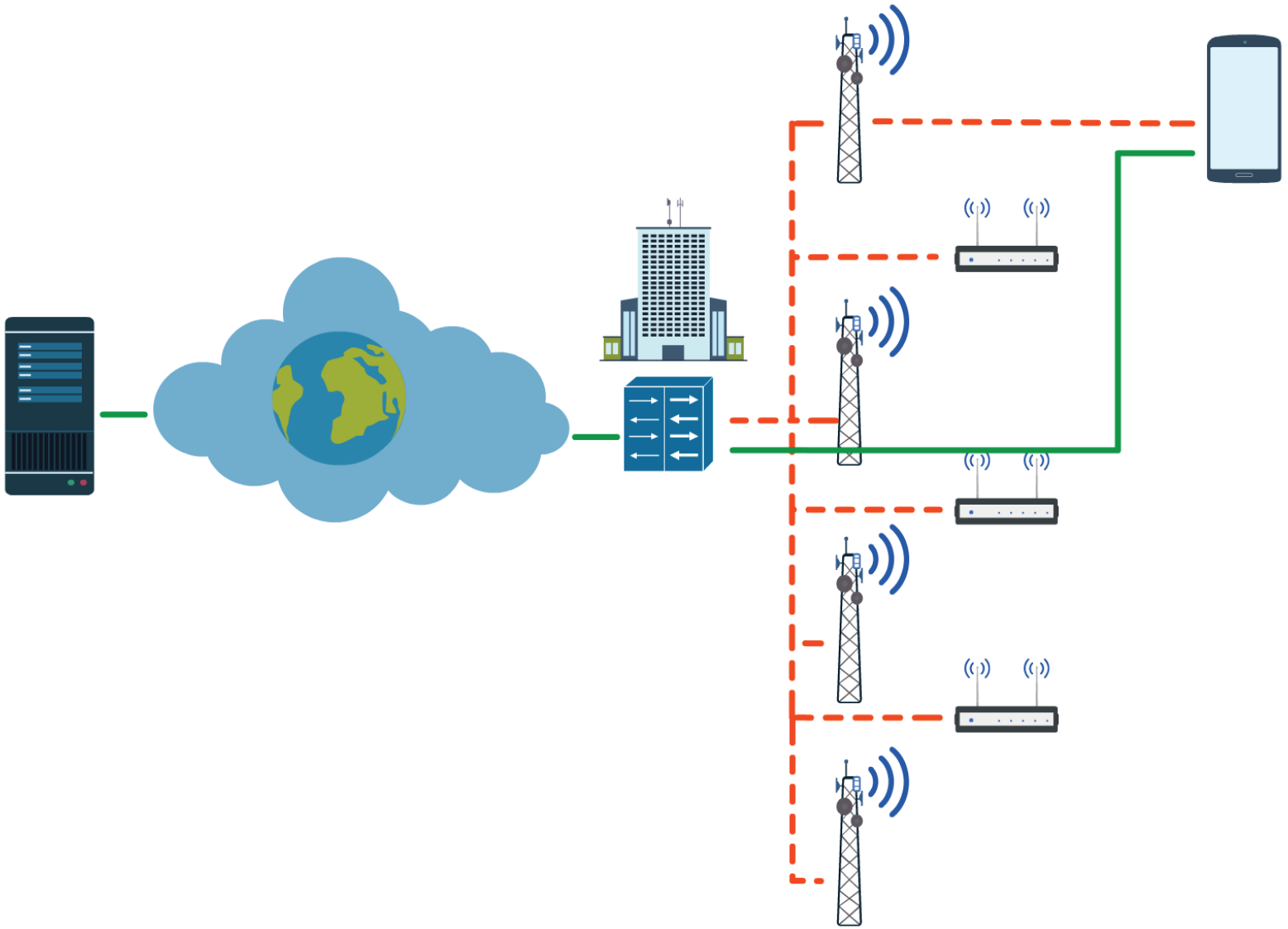
Emerging trend in 5G wireless access



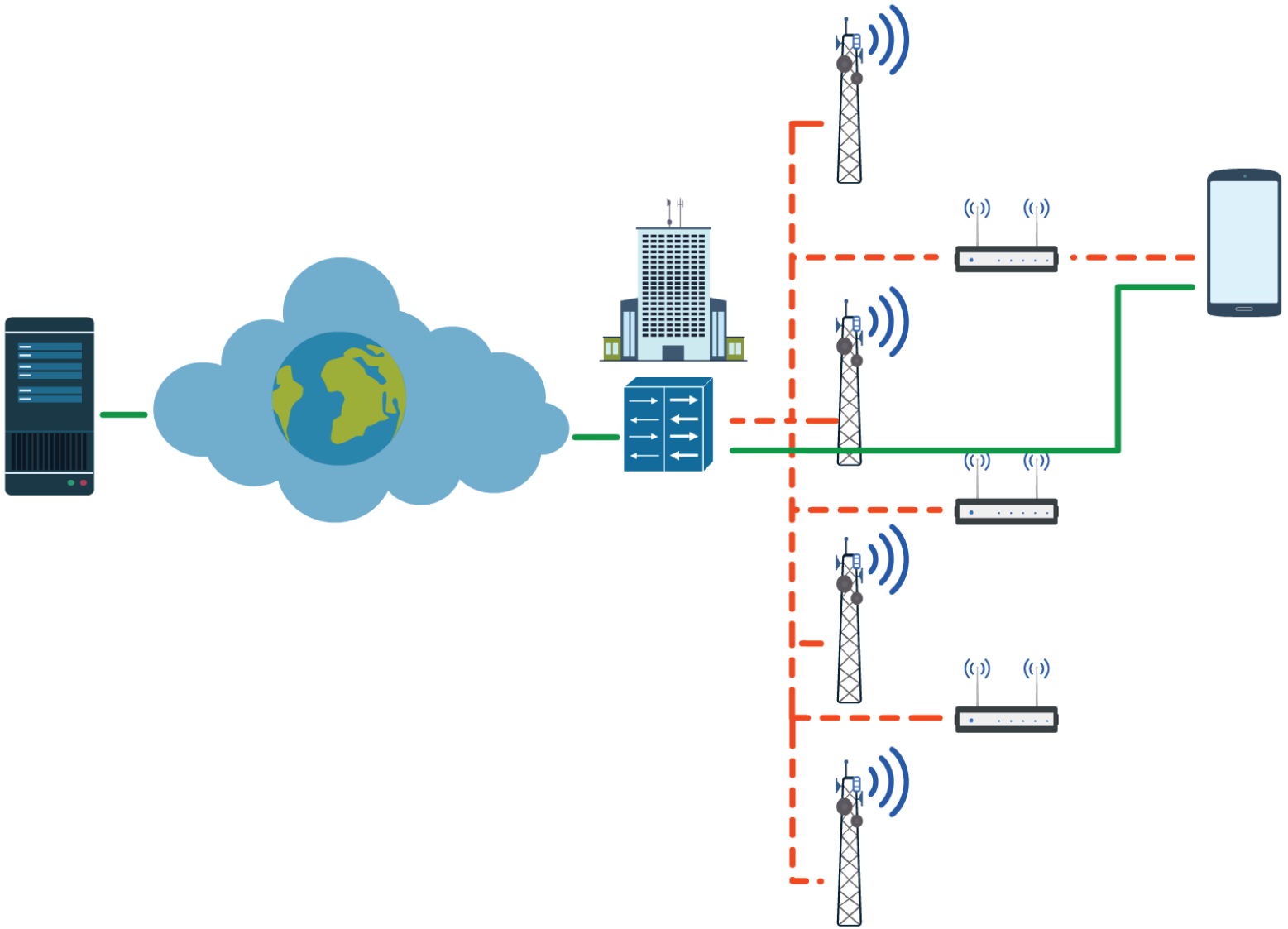
Emerging trend in 5G wireless access



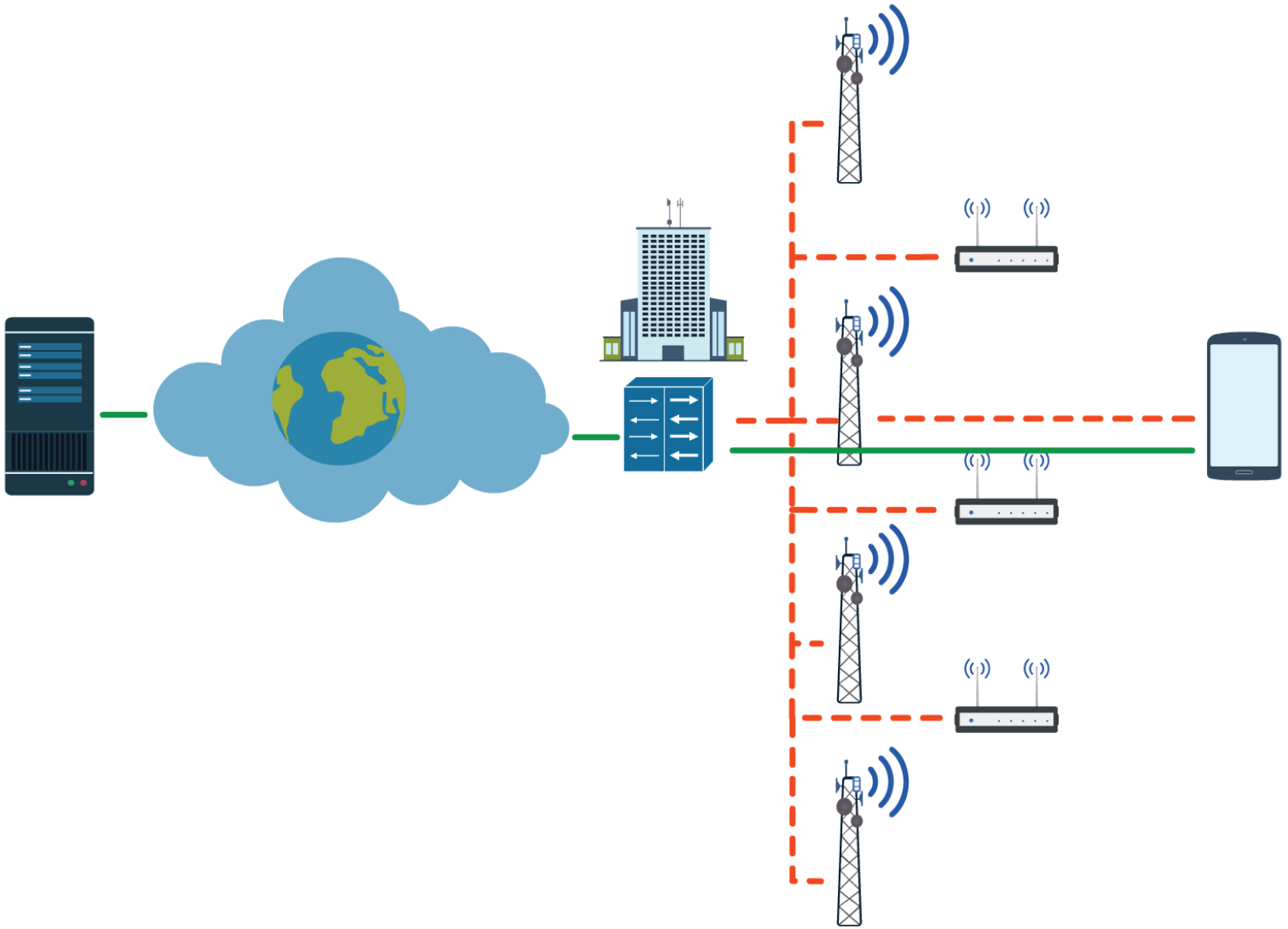
Emerging trend in 5G wireless access



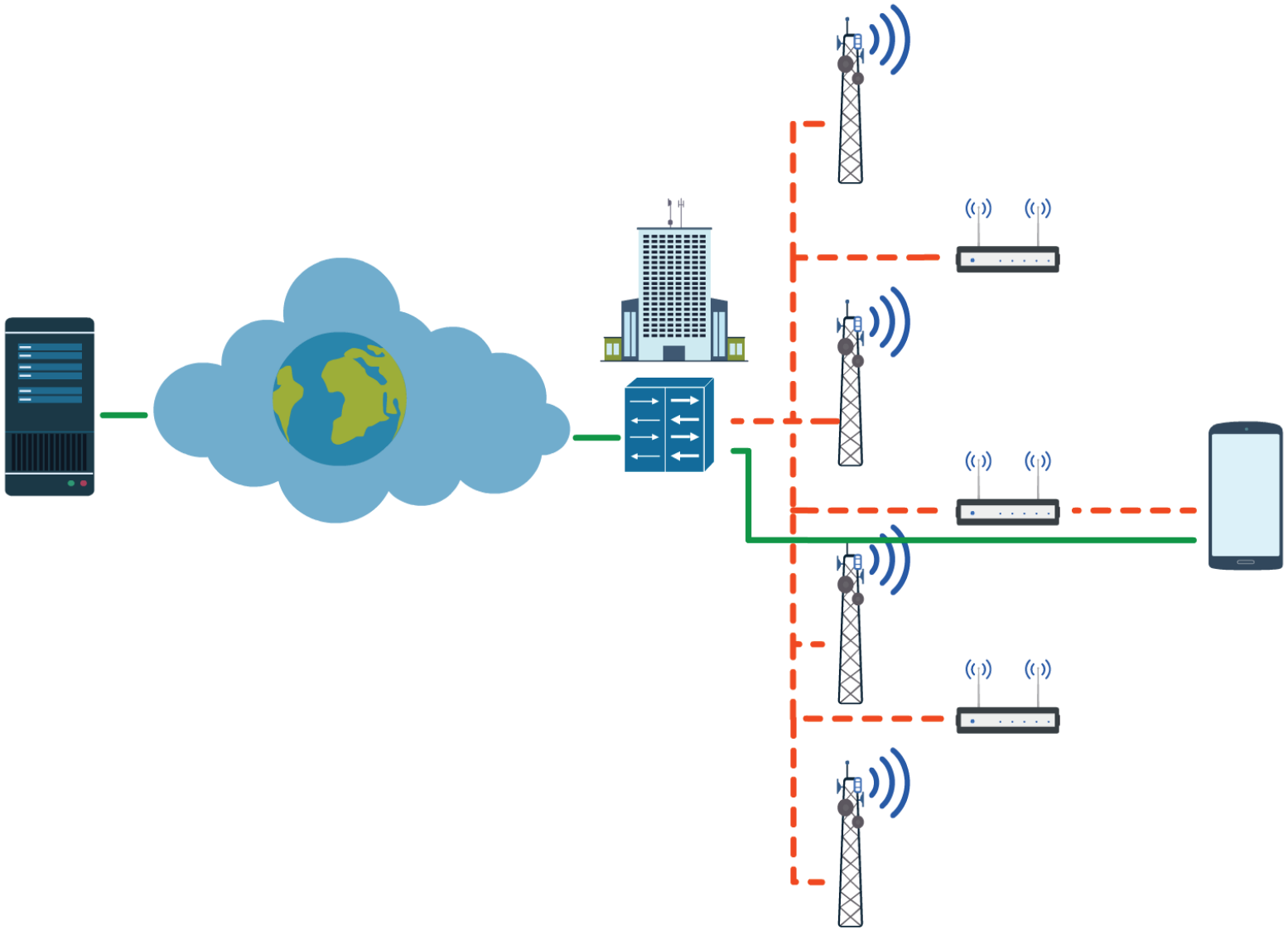
Emerging trend in 5G wireless access



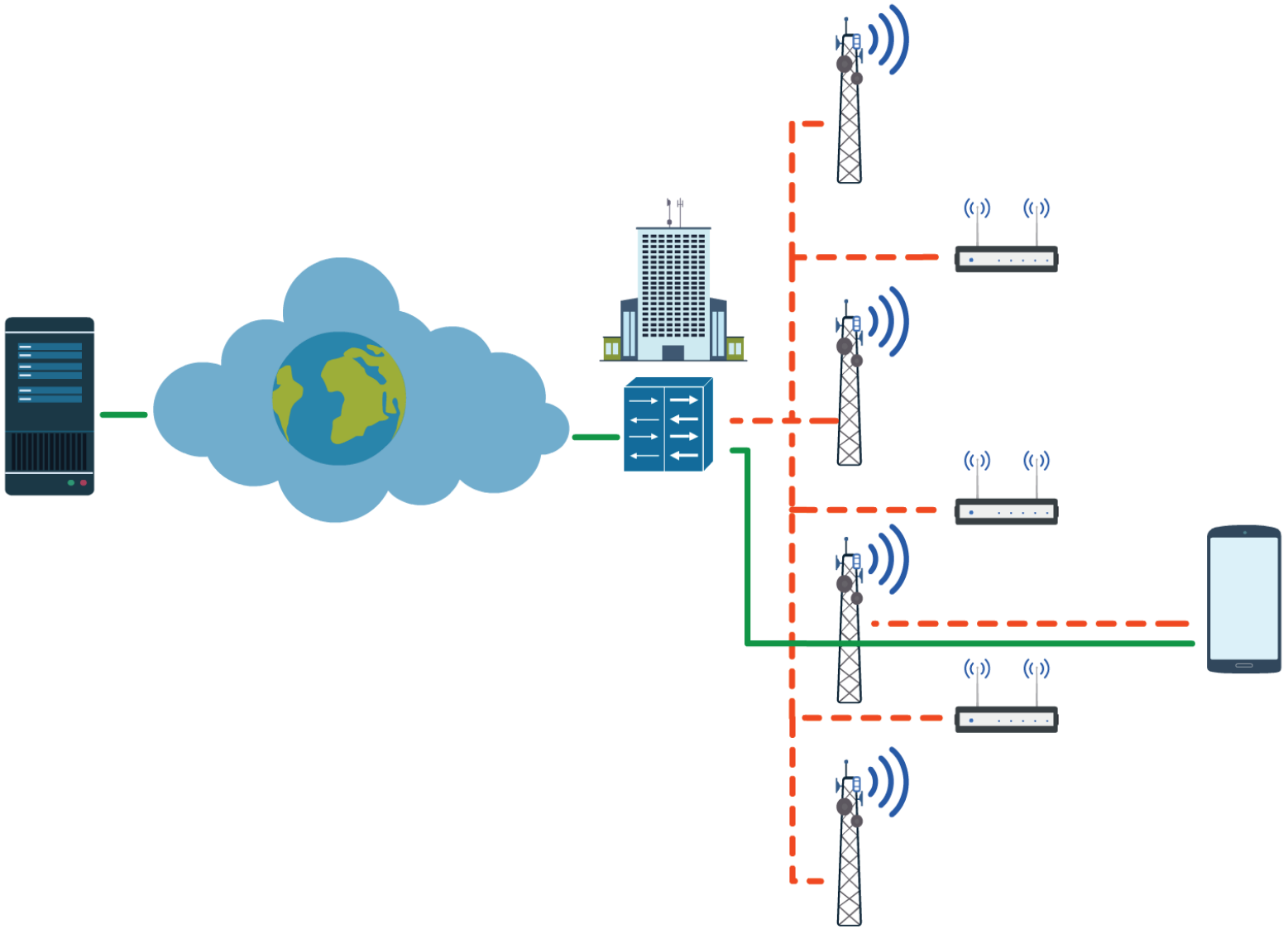
Emerging trend in 5G wireless access



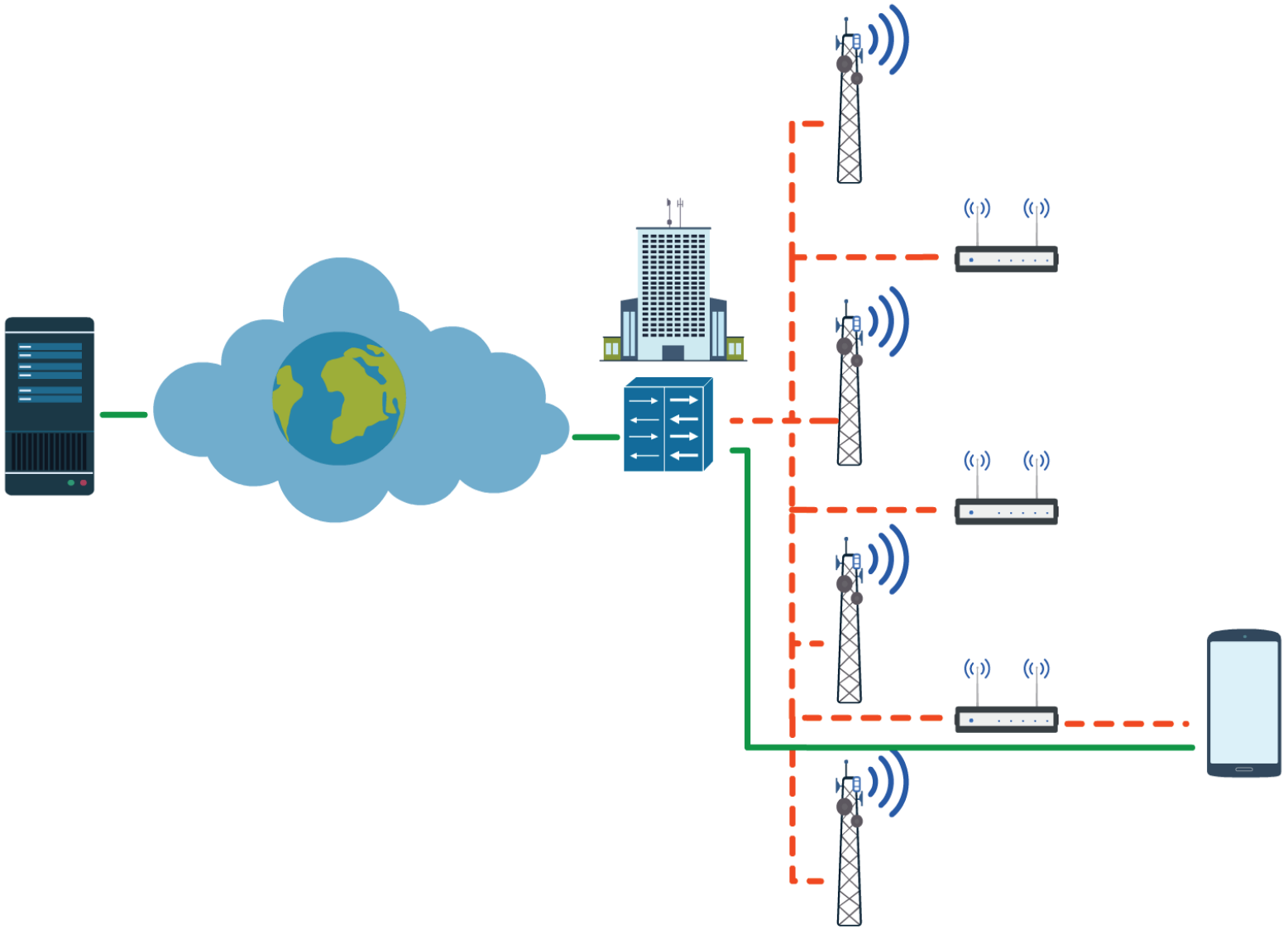
Emerging trend in 5G wireless access



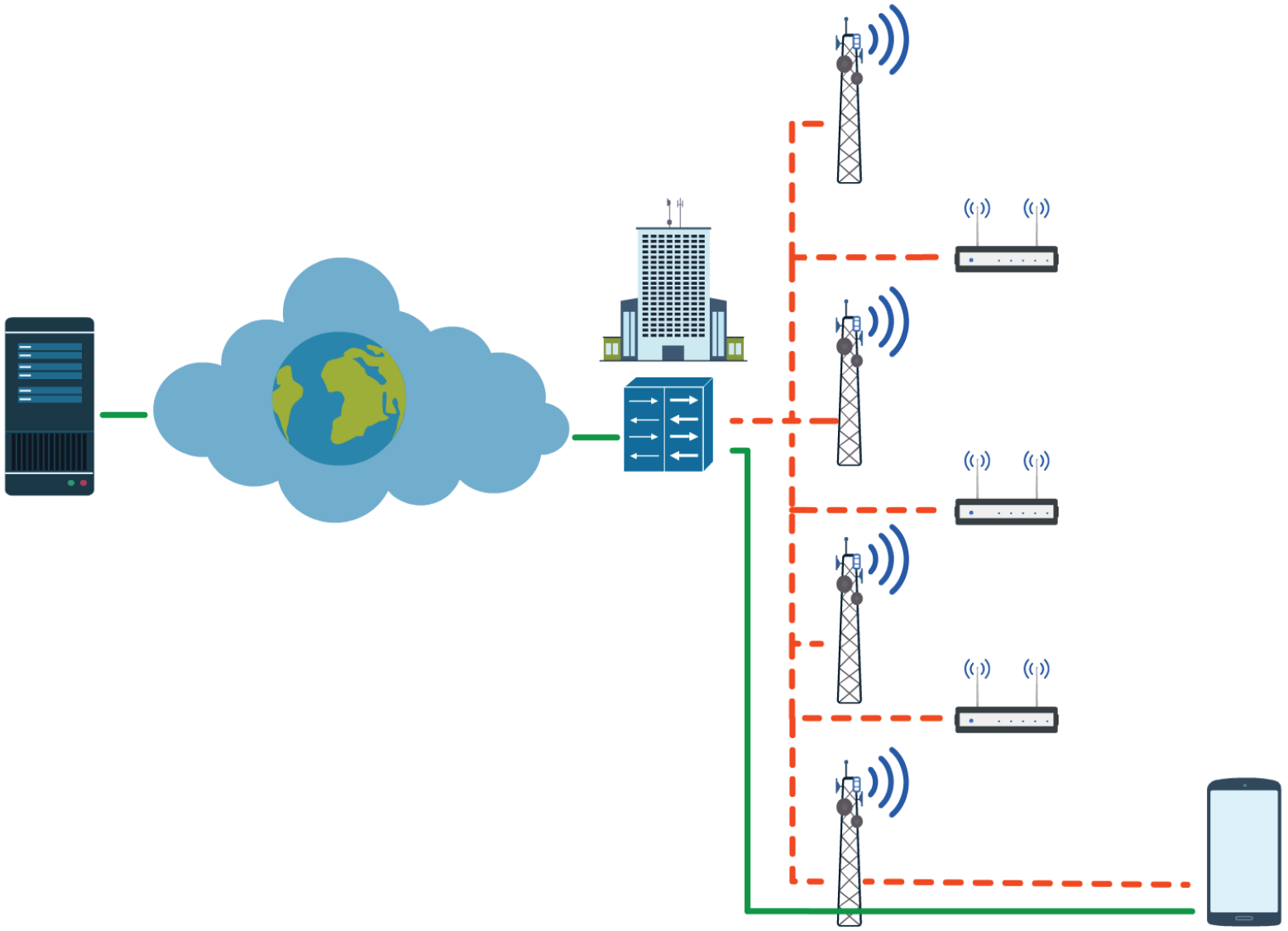
Emerging trend in 5G wireless access



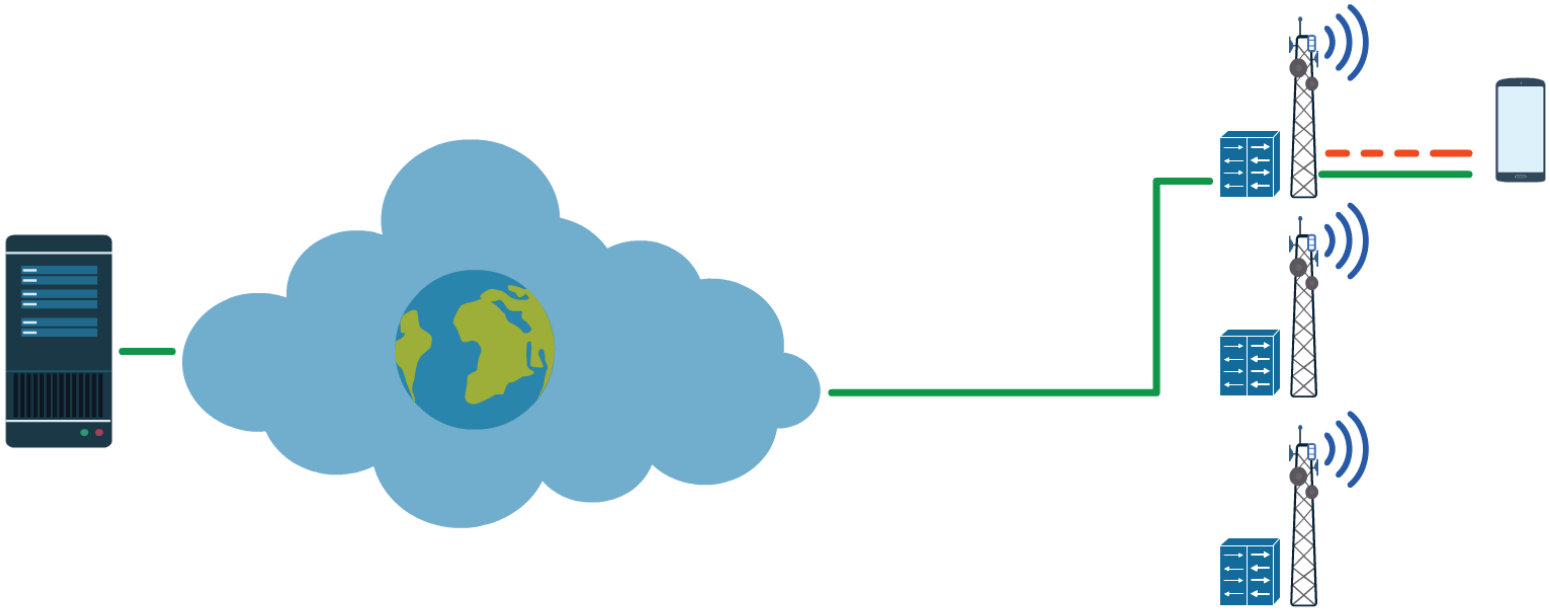
Emerging trend in 5G wireless access



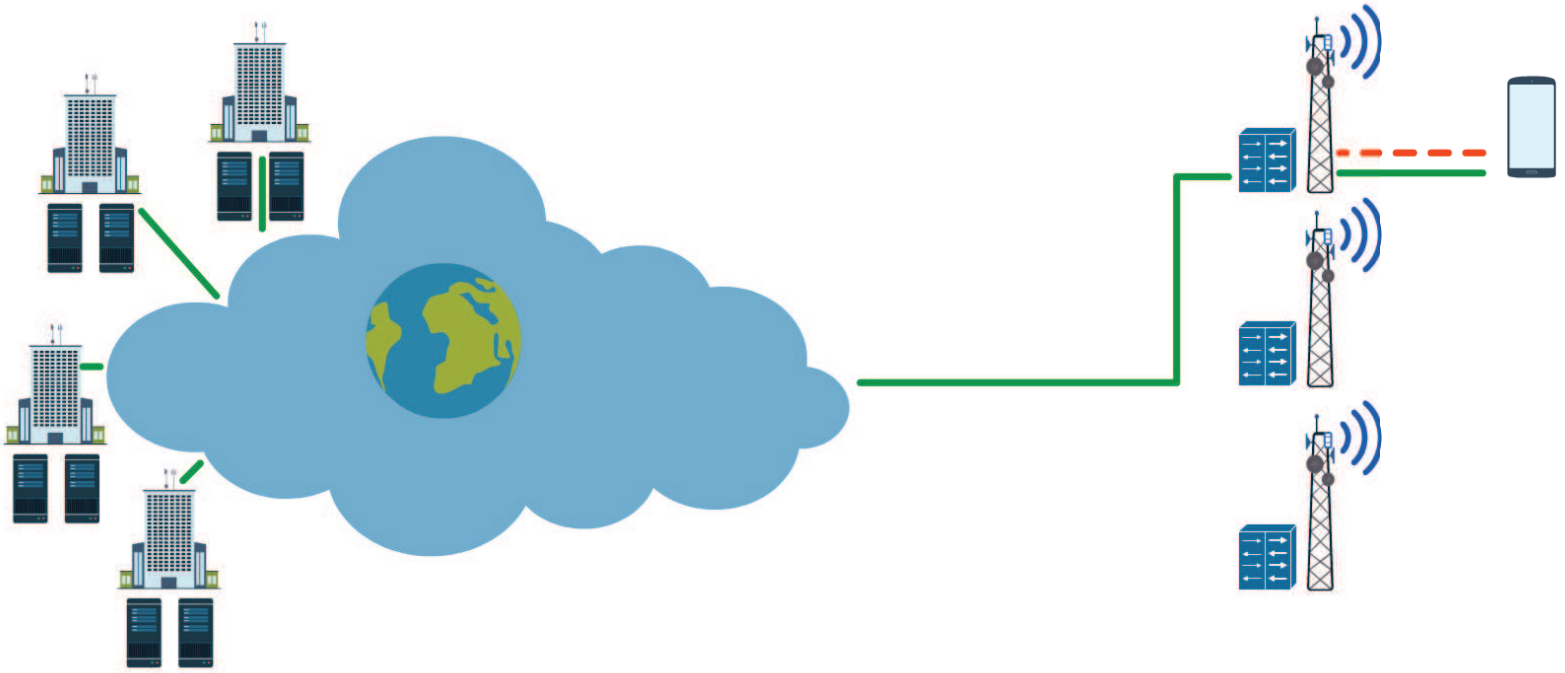
Emerging trend in 5G wireless access



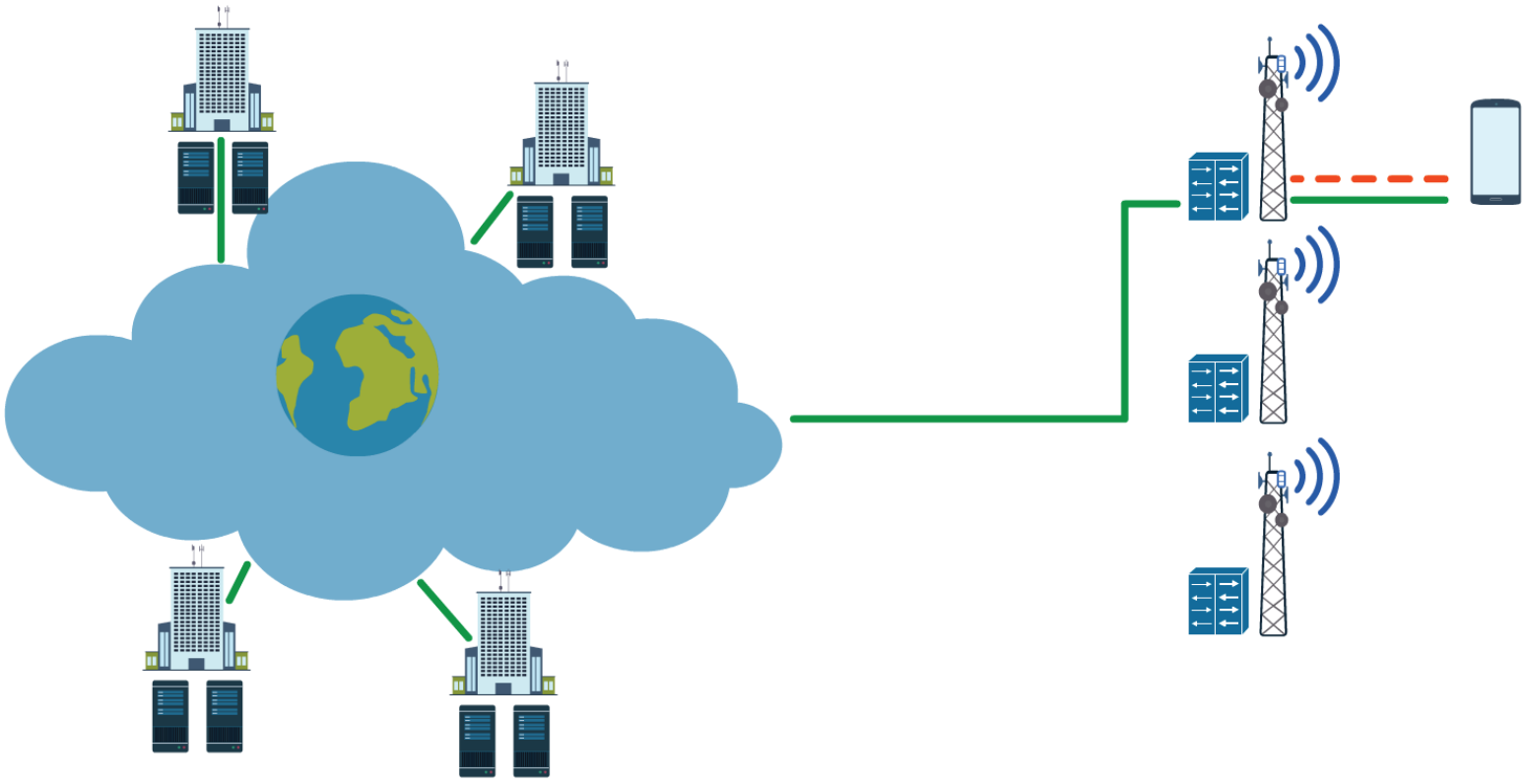
Emerging trend in content delivery



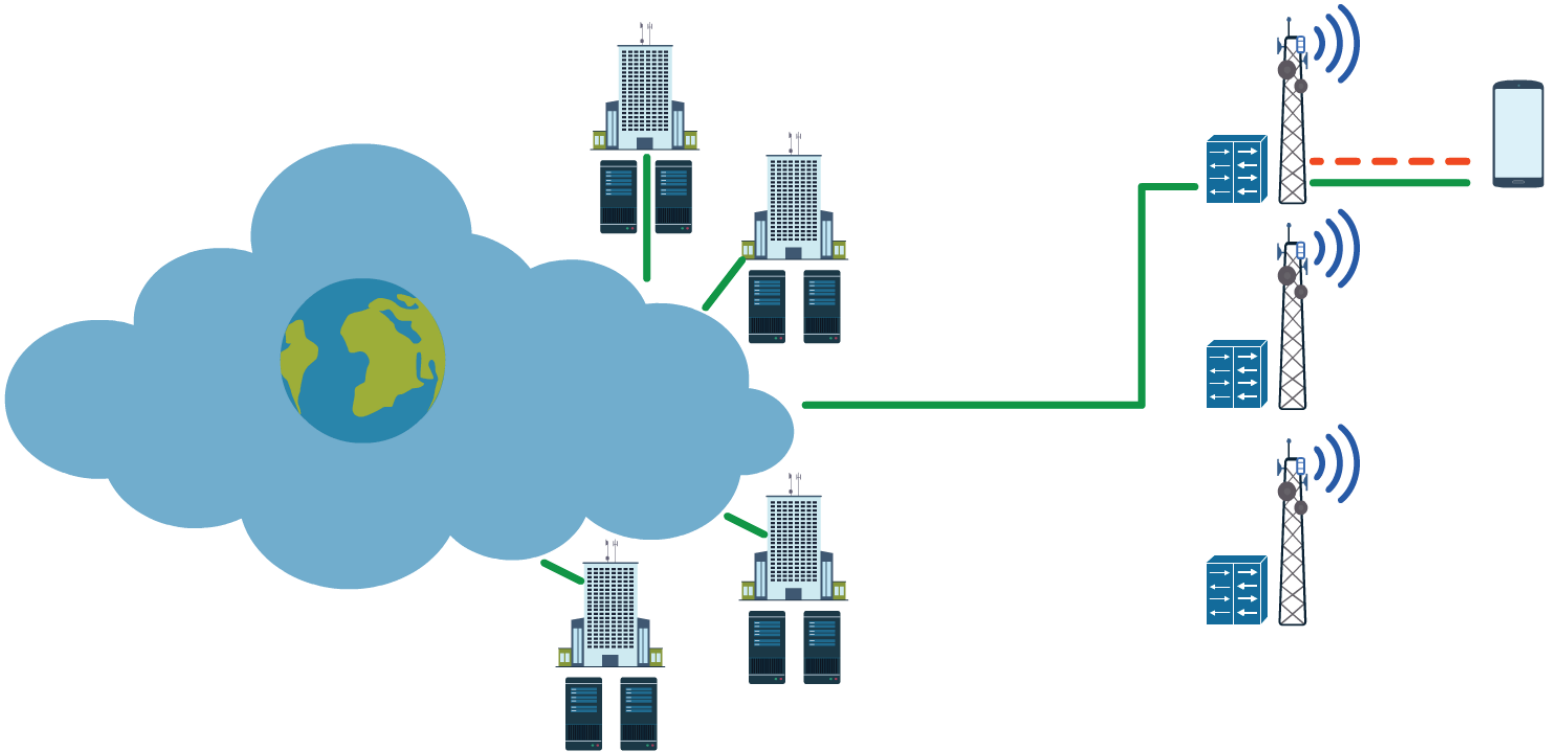
Emerging trend in content delivery



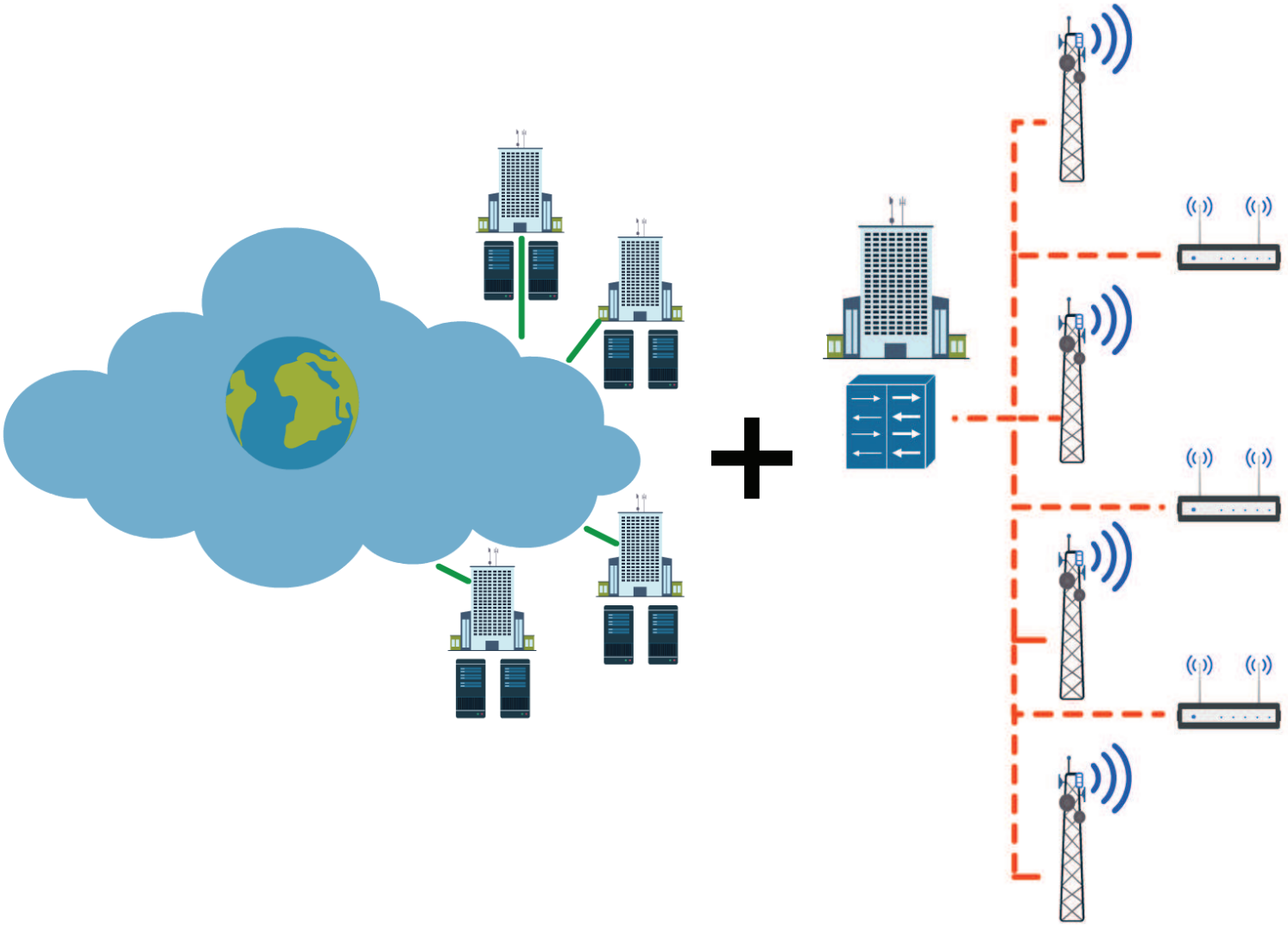
Emerging trend in content delivery



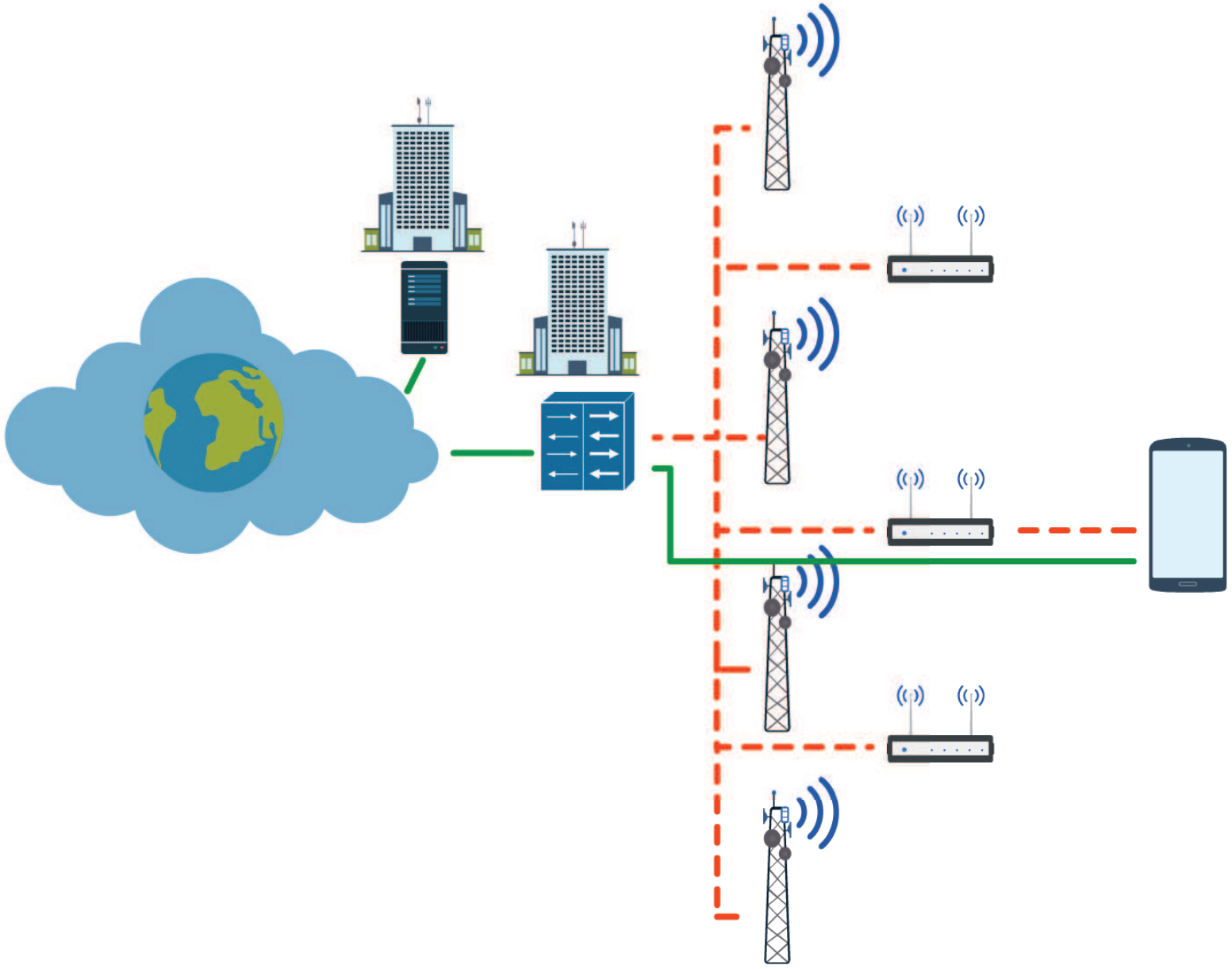
Emerging trend in content delivery



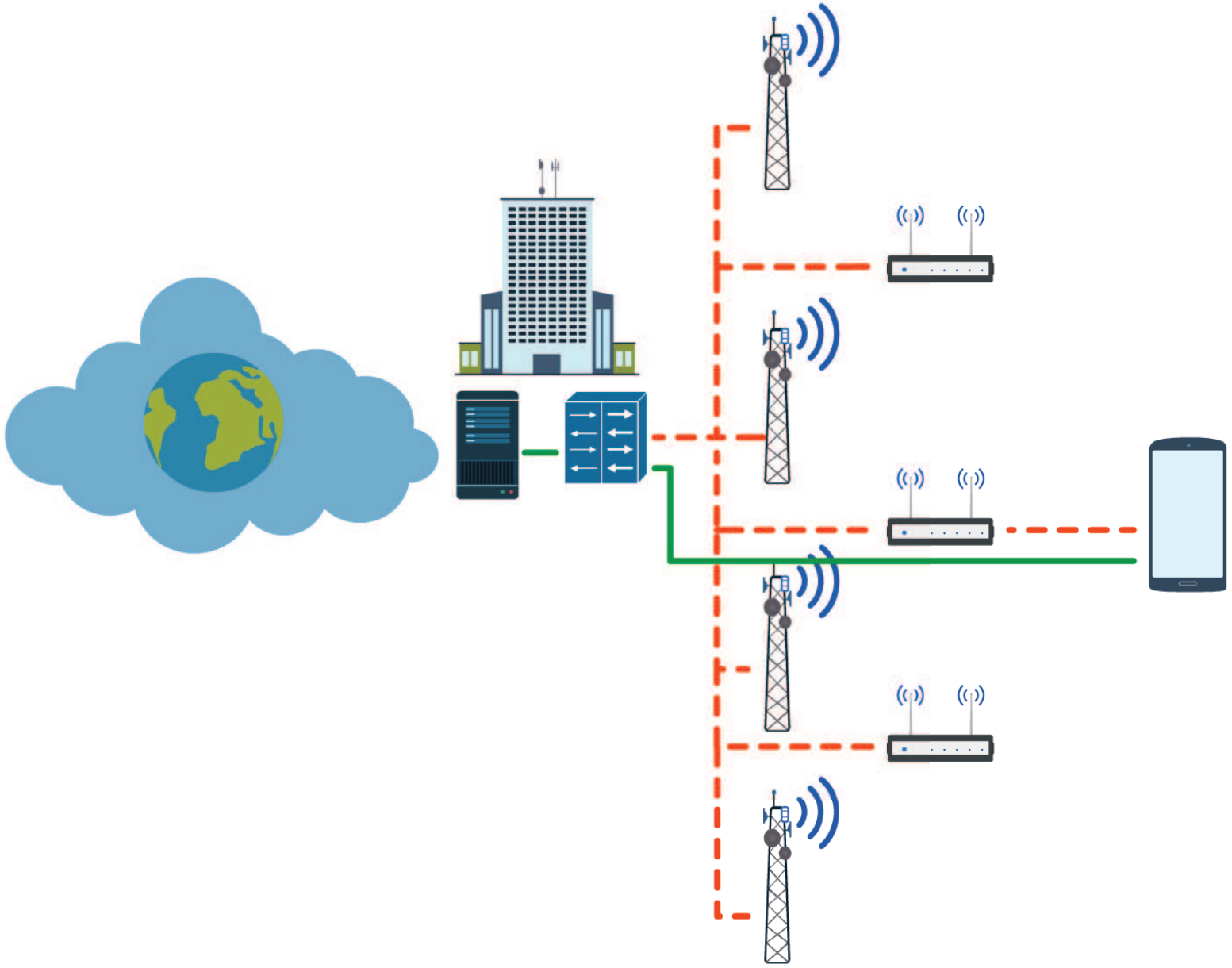
Opportunity



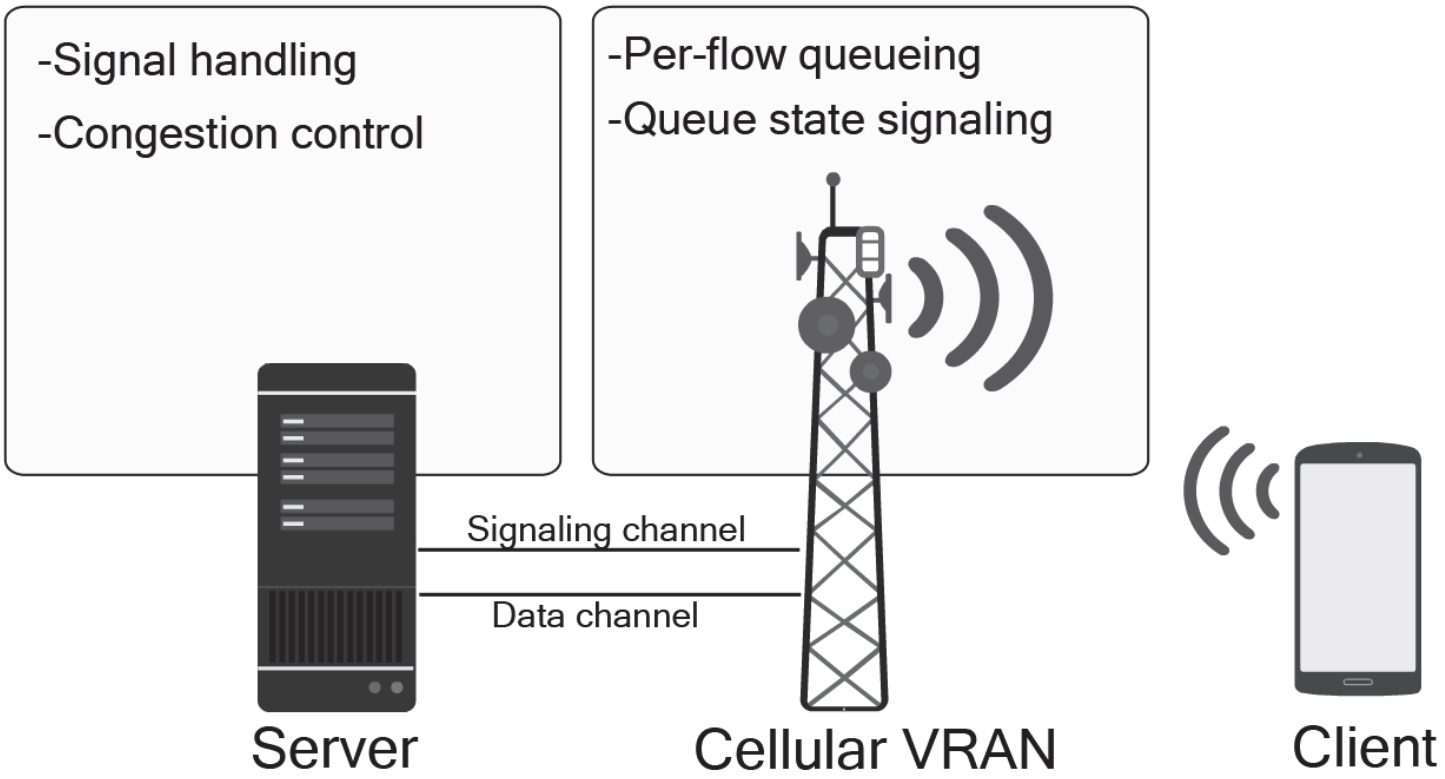
Opportunity



Opportunity



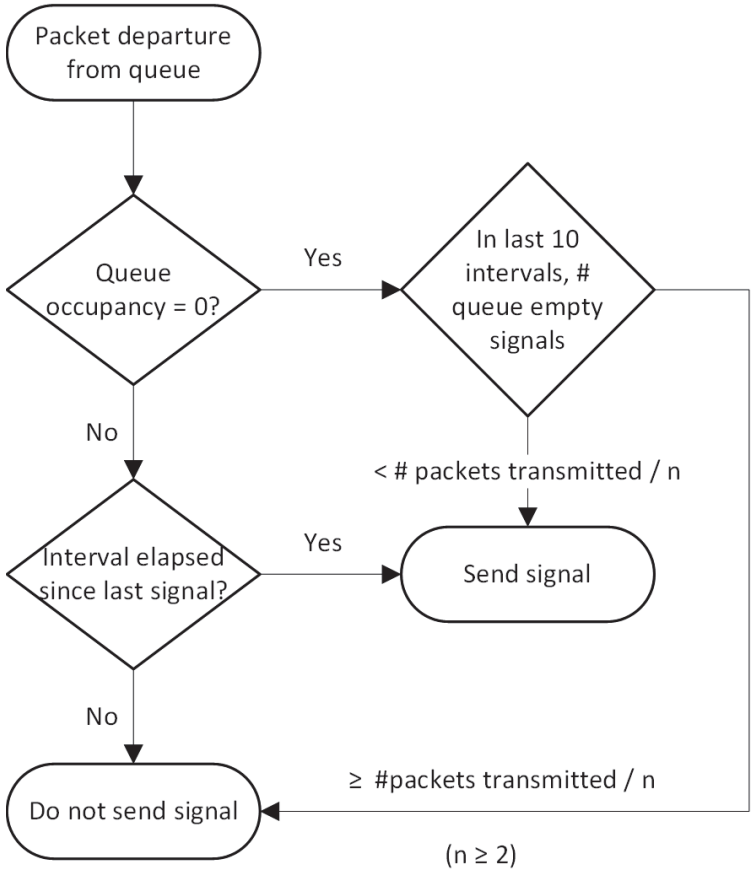
LCTCP Architecture



Provide per-flow queue status information to the server over an out-of-band channel to allow the application server to optimize for a specific utilization vs. delay trade-off.

Network Components

- Per-flow queueing
- Send queue state signals:
  - Periodic queue status signals
  - Immediate queue empty signals
  - Deducing queue balance



Server Components

- Subscribe to queue updates
- Handle signals and identify corresponding flows
- Perform congestion control:
  - Triggered only by queue signals
  - Achieve desired utilization vs. delay trade off
  - Avoid unnecessary queueing

Algorithm 1: Congestion control algorithm handling a queue update signal.

Input: queue occupancy from signal $< p >$
Data: target queue balance $< target >$
Data: signaling interval $< interval >$
Data: desired throughput vs. delay tradeoff $< util >$
Data: last target update timestamp $< last\_update >$
Data: congestion window [pkt] $< cwnd >$
Data: measured round-trip time $< rtt >$
Data: current time $< now >$

```
1  error = target − p;
2  if p == 0 then
3  |   random = random_int() % 1024;
4  |   if random ≤ util then
5  |   |   cwnd = cwnd + 1;
6  |   |   target = max(1.05 ∗ target, target + 1);
7  |   |   last_update = now;
8  |   end
9  end
10 else if target > 2 AND |error| > 0.05 ∗ target then
11 |   cwnd = max(cwnd + error ∗ interval/rtt, 2);
12 end
13 if now − last_update > 10 ∗ rtt then
14 |   target = max(0.95 ∗ target, 2);
15 |   last_update = now;
16 end
```

Congestion Control

Algorithm 1: Congestion control algorithm handling a queue update signal.

Input: queue occupancy from signal $< p >$
Data: target queue balance $< target >$
Data: signaling interval $< interval >$
Data: desired throughput vs. delay tradeoff $< util >$
Data: last target update timestamp $< last\_update >$
Data: congestion window [pkt] $< cwnd >$
Data: measured round-trip time $< rtt >$
Data: current time $< now >$

```
1  error = target − p;
2  if p == 0 then
3  |   random = random_int() % 1024;
4  |   if random ≤ util then
5  |   |   cwnd = cwnd + 1;
6  |   |   target = max(1.05 ∗ target, target + 1);
7  |   |   last_update = now;
8  |   end
9  end
10 else if target > 2 AND |error| > 0.05 ∗ target then
11 |   cwnd = max(cwnd + error ∗ interval/rtt, 2);
12 end
13 if now − last_update > 10 ∗ rtt then
14 |   target = max(0.95 ∗ target, 2);
15 |   last_update = now;
16 end
```

Congestion Control

Algorithm 1: Congestion control algorithm handling a queue update signal.

Input: queue occupancy from signal $< p >$
Data: target queue balance $< target >$
Data: signaling interval $< interval >$
Data: desired throughput vs. delay tradeoff $< util >$
Data: last target update timestamp $< last\_update >$
Data: congestion window [pkt] $< cwnd >$
Data: measured round-trip time $< rtt >$
Data: current time $< now >$

```
1  error = target − p;
2  if p == 0 then
3  |   random = random_int() % 1024;
4  |   if random ≤ util then
5  |   |   cwnd = cwnd + 1;
6  |   |   target = max(1.05 ∗ target, target + 1);
7  |   |   last_update = now;
8  |   end
9  end
10 else if target > 2 AND |error| > 0.05 ∗ target then
11 |   cwnd = max(cwnd + error ∗ interval/rtt, 2);
12 end
13 if now − last_update > 10 ∗ rtt then
14 |   target = max(0.95 ∗ target, 2);
15 |   last_update = now;
16 end
```

Negative queue balance

Positive queue balance

UT DALLAS
The University of Texas at Dallas

Congestion Control

Algorithm 1: Congestion control algorithm handling a queue update signal.

Input: queue occupancy from signal $< p >$
Data: target queue balance $< target >$
Data: signaling interval $< interval >$
Data: desired throughput vs. delay tradeoff $< util >$
Data: last target update timestamp $< last\_update >$
Data: congestion window [pkt] $< cwnd >$
Data: measured round-trip time $< rtt >$
Data: current time $< now >$

```
1  error = target − p;
2  if p == 0 then
3  |  random = random_int() % 1024;
4  |  if random ≤ util then
5  |  |  cwnd = cwnd + 1;
6  |  |  target = max(1.05 ∗ target, target + 1);
7  |  |  last_update = now;
8  |  end
9  end
10 else if target > 2 AND |error| > 0.05 ∗ target then
11 |  cwnd = max(cwnd + error ∗ interval/rtt, 2);
12 end
13 if now − last_update > 10 ∗ rtt then
14 |  target = max(0.95 ∗ target, 2);
15 |  last_update = now;
16 end
```

Link unstable

Link stable

Negative queue balance

Positive queue balance

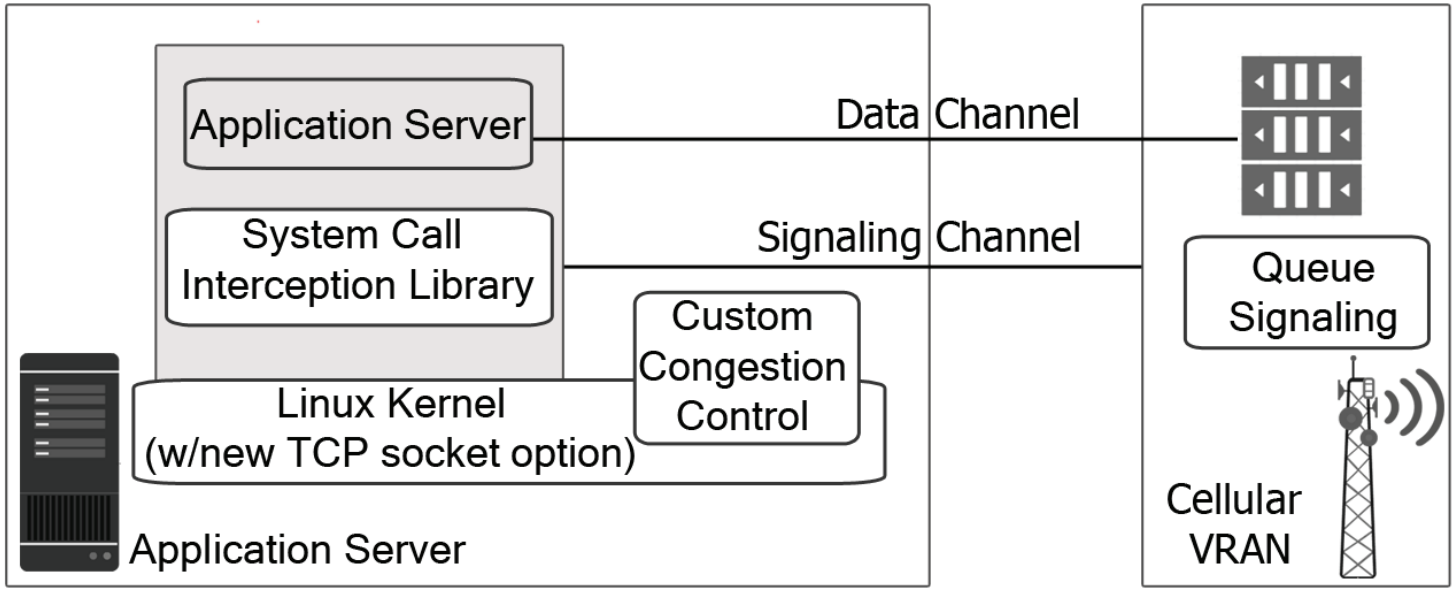
Signaling Overhead

- Queue state update signal $<$ TCP ACK-only packet
- Lower bound: $\frac{1}{\text{signaling\_interval}}$
- Upper bound: $\frac{\text{queue\_packet\_drain\_rate}}{n}$

$\Big\}$ signals per second

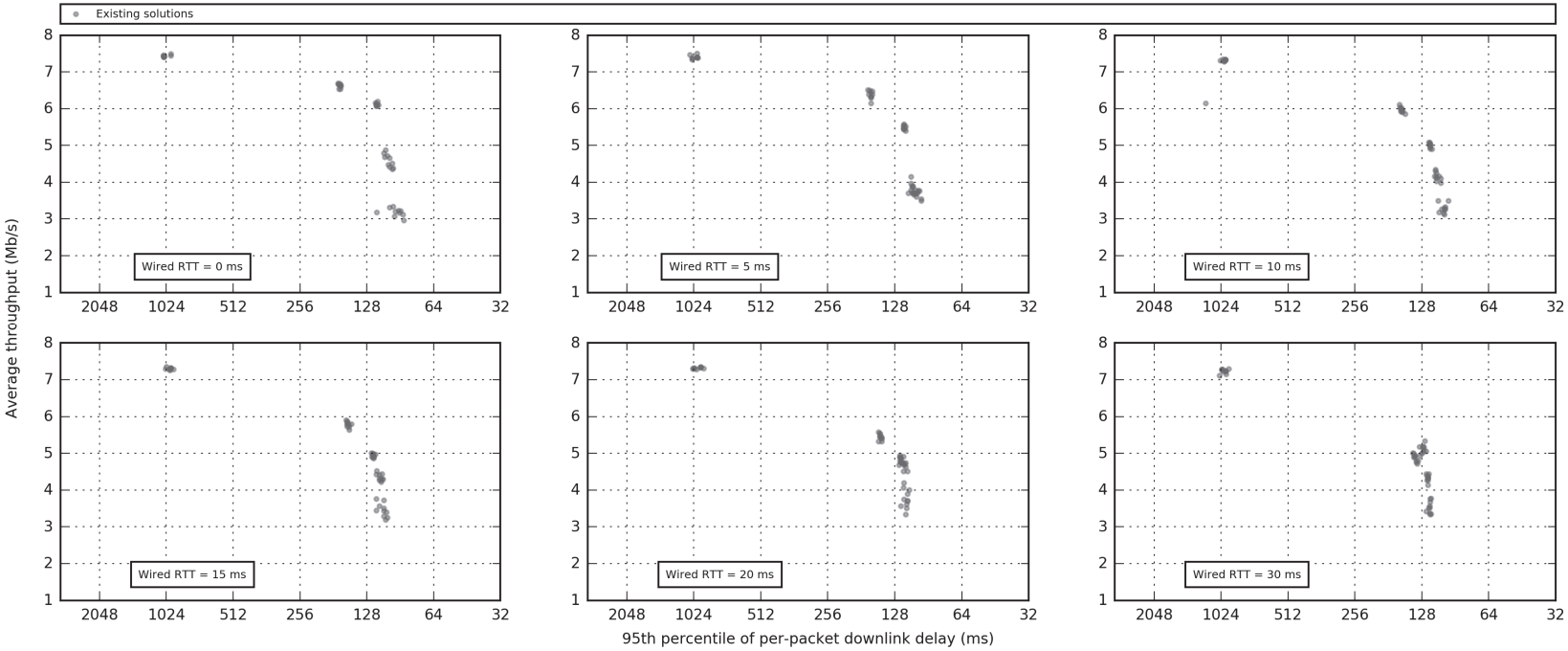
But what do we gain?

Prototype

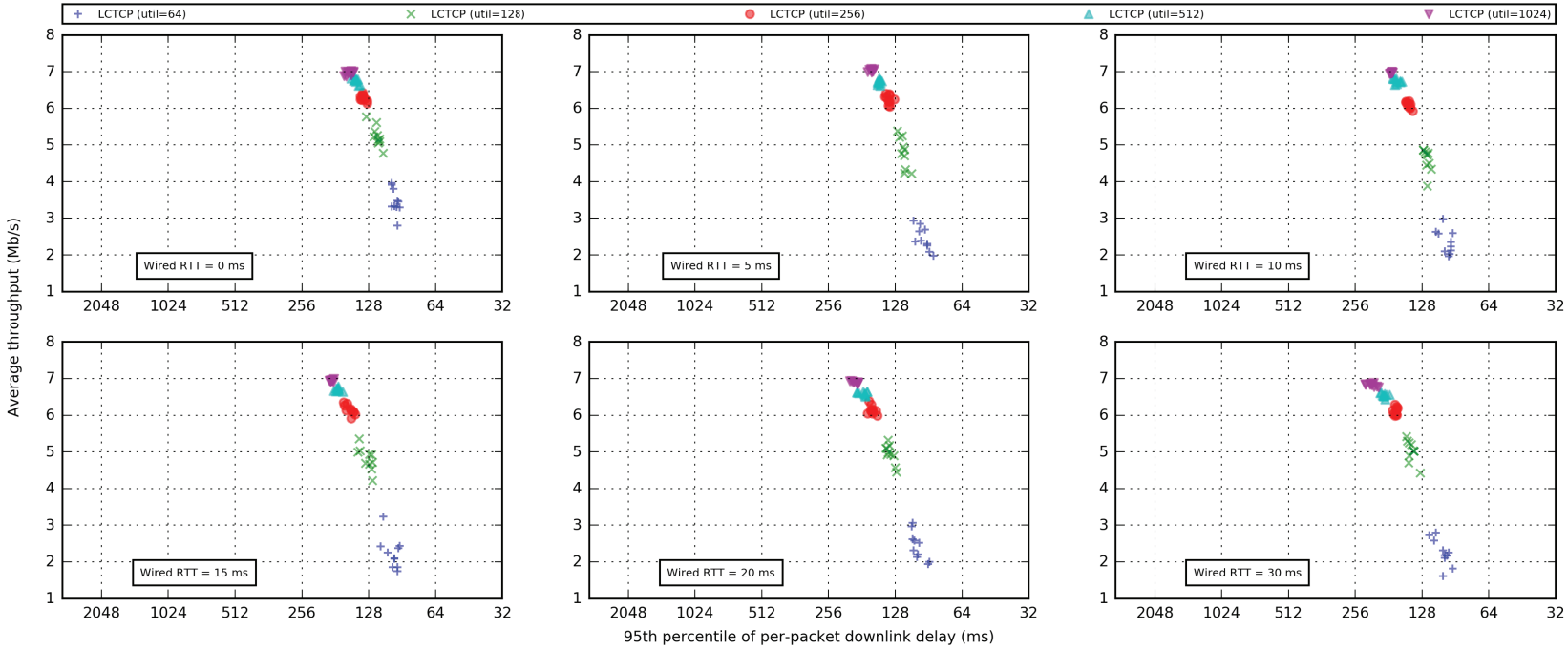


- Linux
- Mahimahi
- LTE Driving traces
- UDP signaling

Performance With Varying RTT

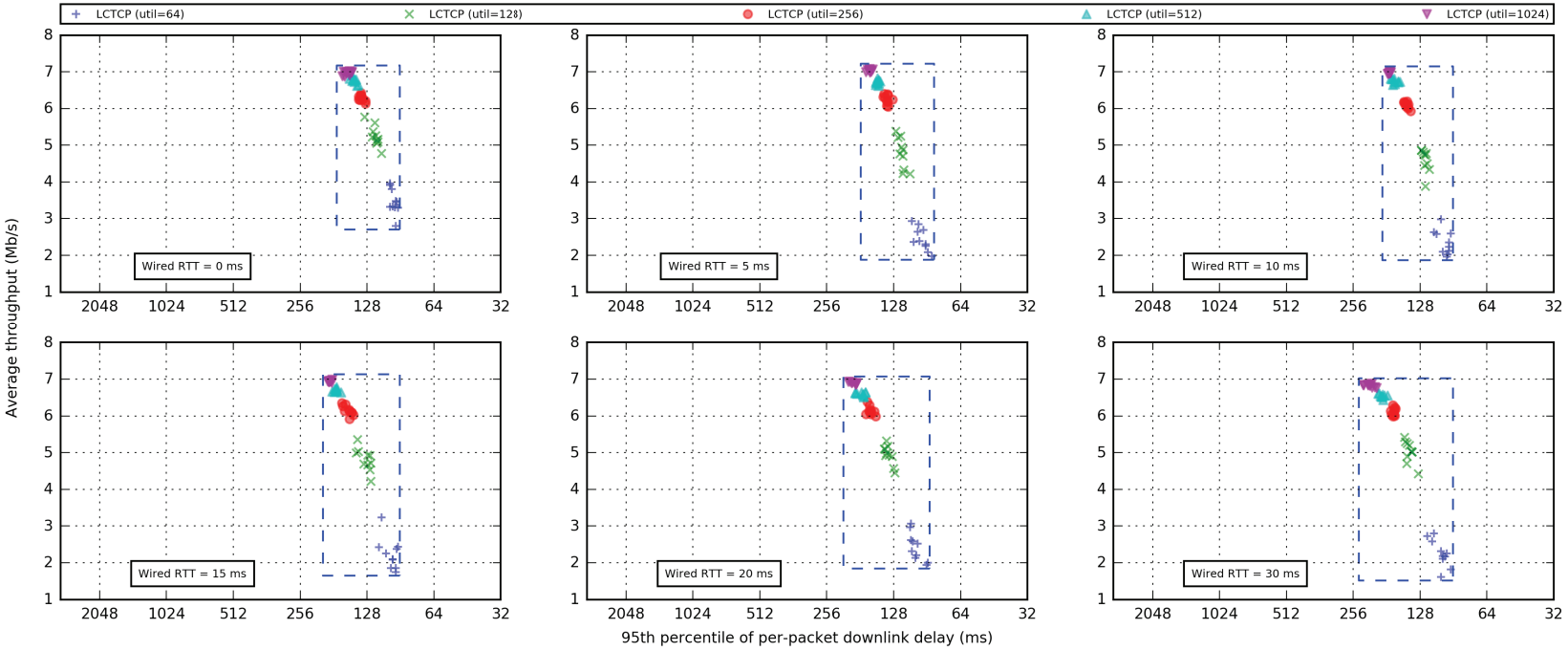


Performance With Varying RTT



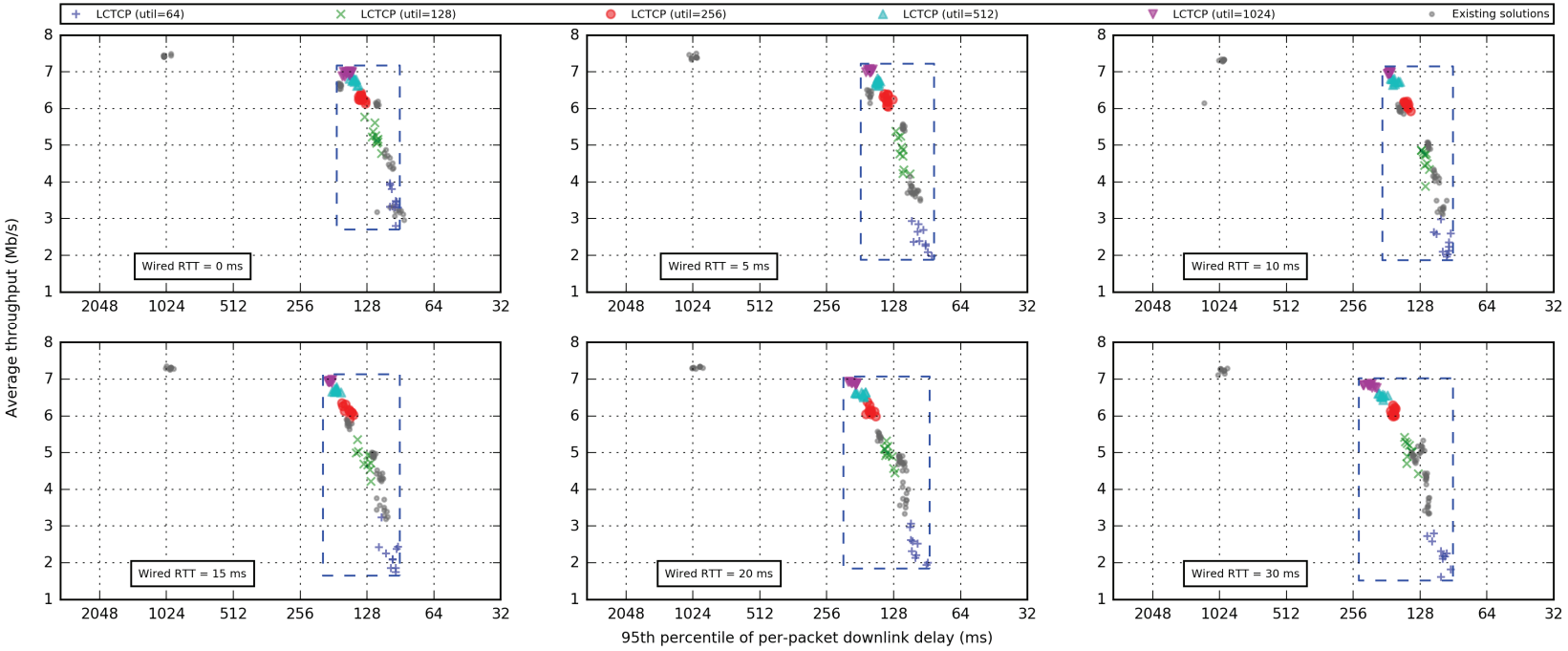
- Covers whole feasible solutions range
- Achieves predictable utilization v. delay tradeoff regardless of RTT
- Properly prioritizes utilization over delay or vv.

Performance With Varying RTT



- Covers whole feasible solutions range
- Achieves predictable utilization v. delay tradeoff regardless of RTT
- Properly prioritizes utilization over delay or vv.

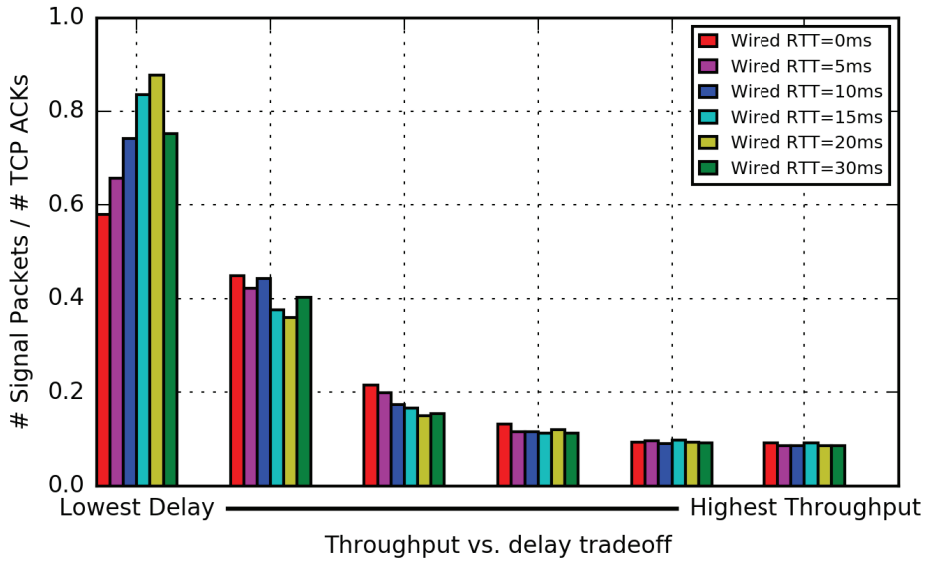
Performance With Varying RTT



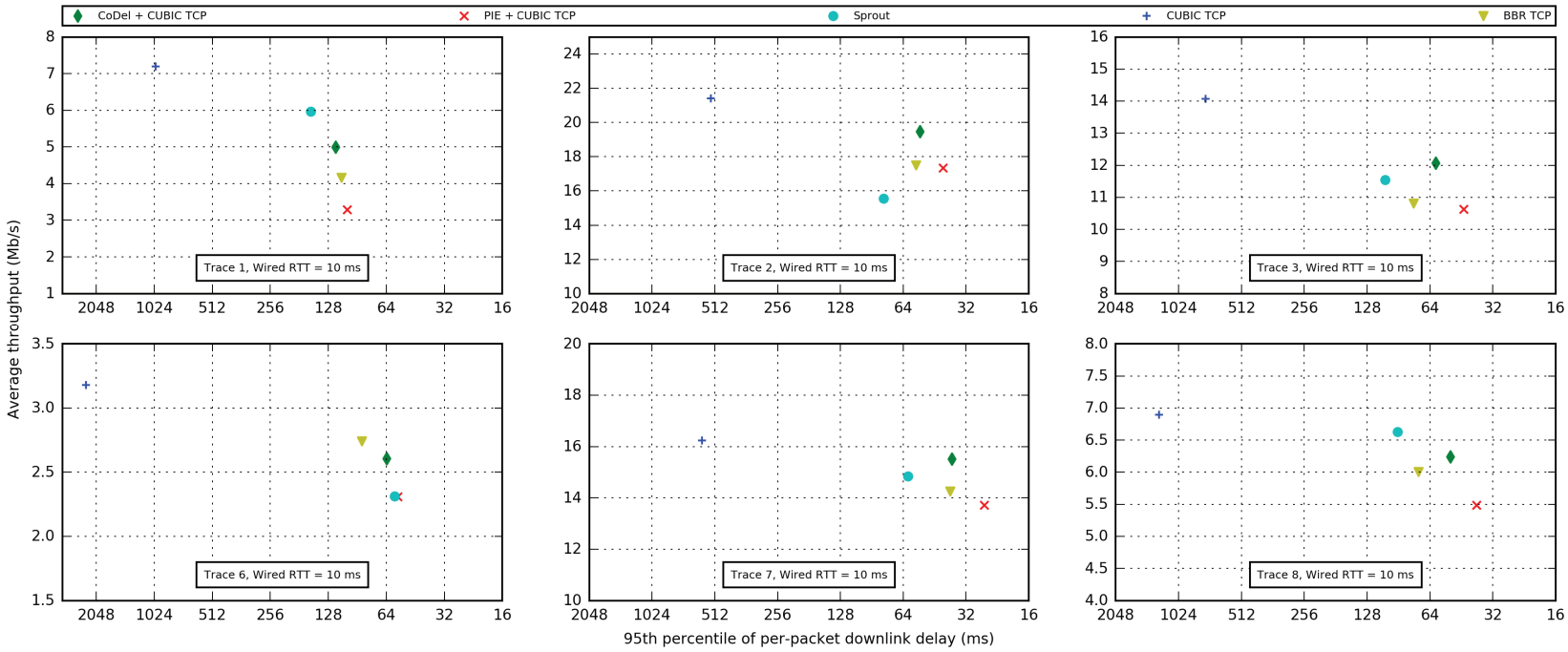
- Covers whole feasible solutions range
- Achieves predictable utilization v. delay tradeoff regardless of RTT
- Properly prioritizes utilization over delay or vv.

Performance With Varying RTT: Overhead

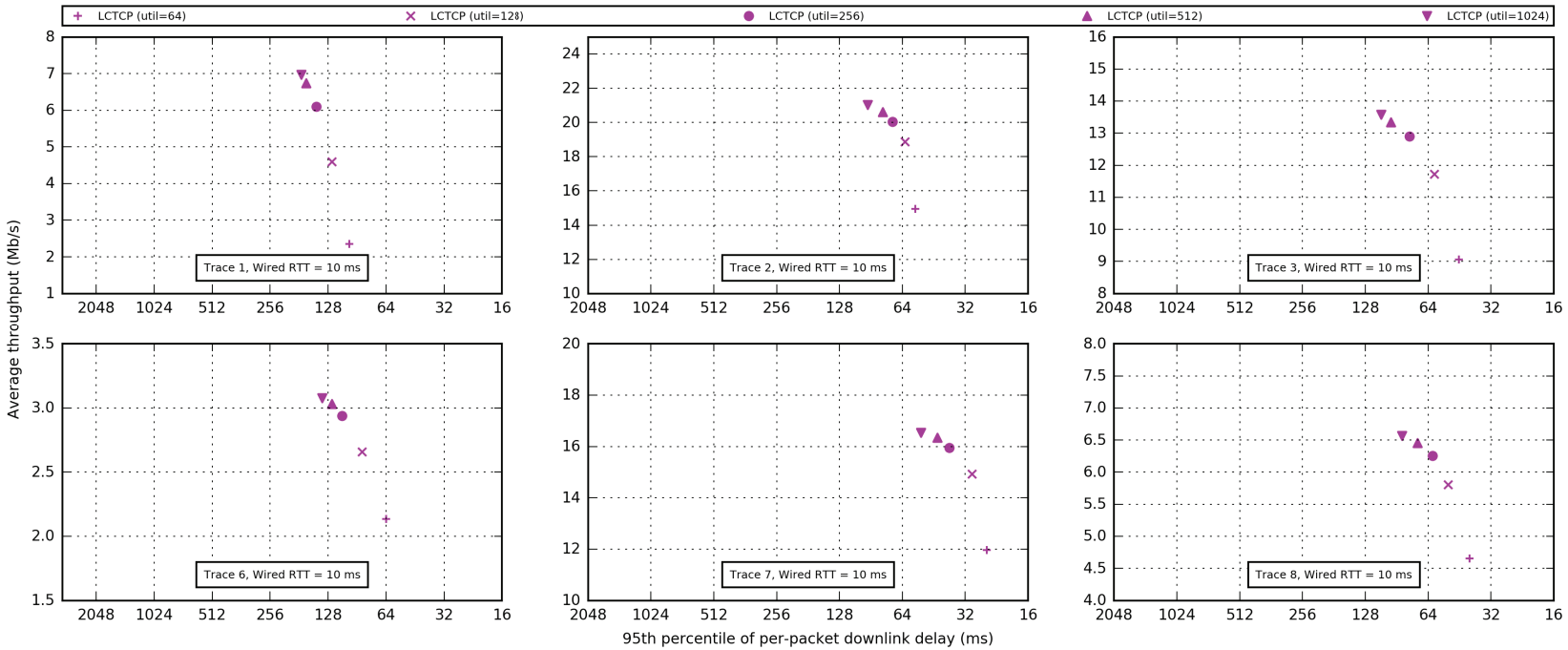
- Fraction of the TCP ACK traffic
- Highest overhead when lowest delay is desired
- Overhead quickly diminishes when utilization is also required



Performance With Other Cellular Traces

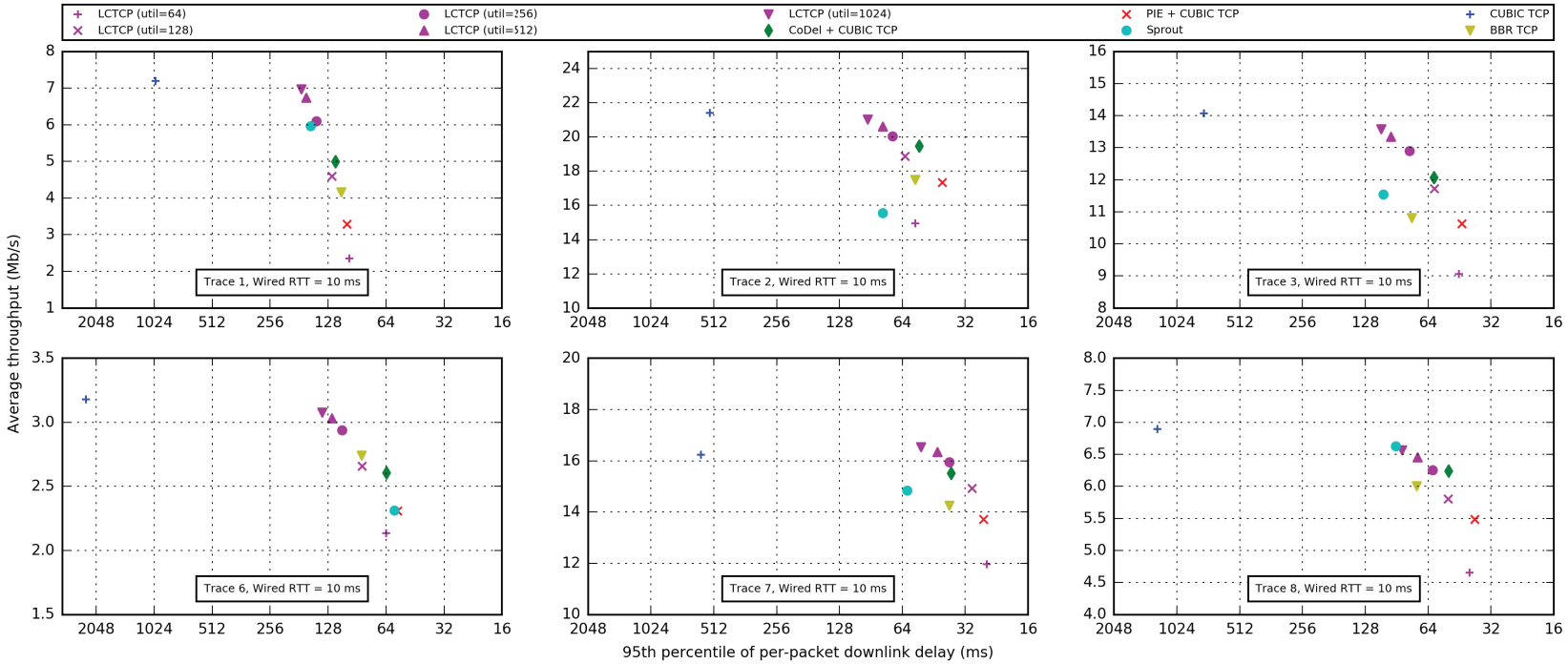


Performance With Other Cellular Traces



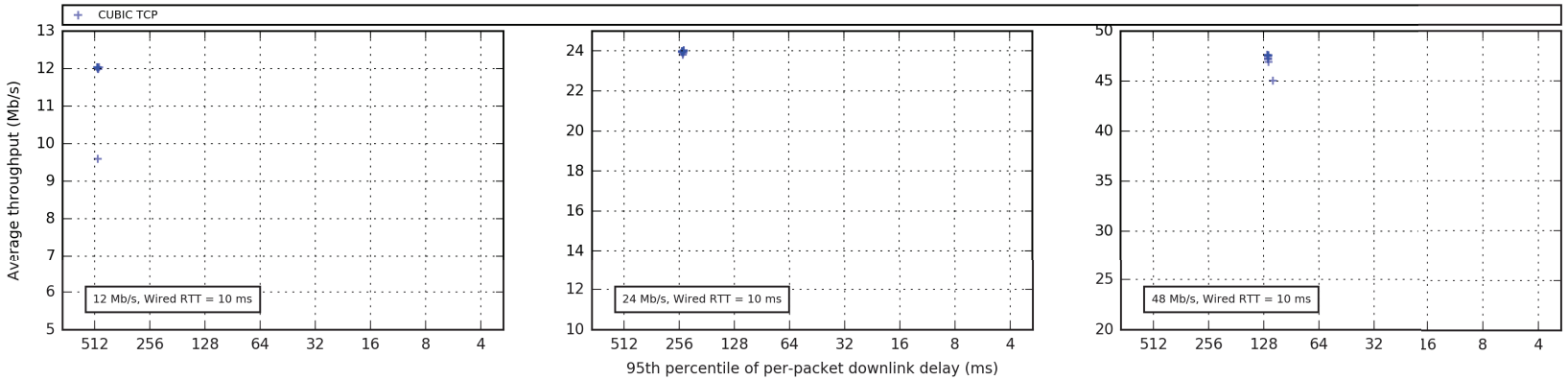
- Existing solutions change trade-off point with each trace
- LCTCP achieves predictible trade off and covers all the trade-off range

Performance With Other Cellular Traces

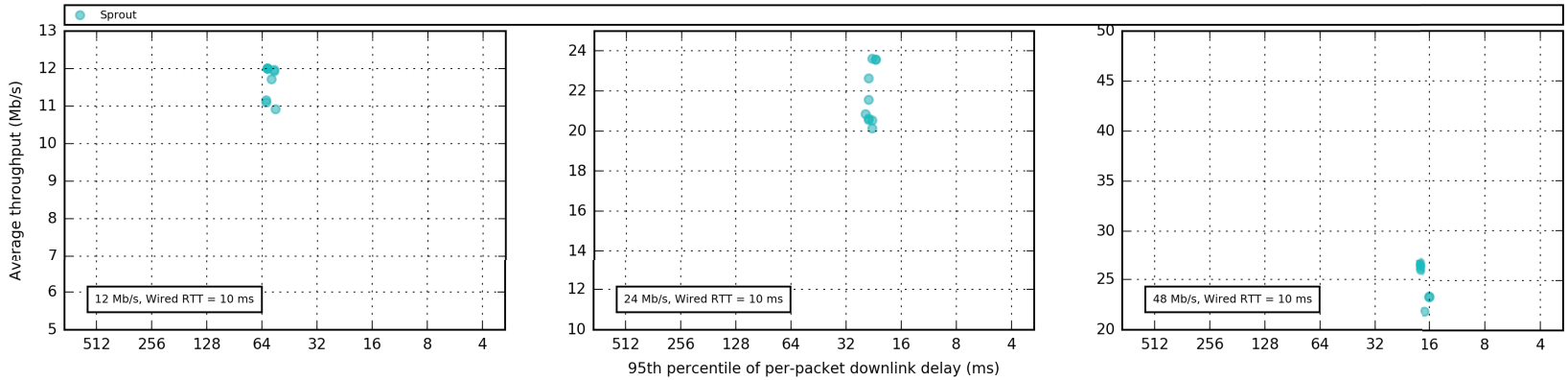


- Existing solutions change trade-off point with each trace
- LCTCP achieves predictible trade off and covers all the trade-off range

Performance With Fixed Bandwidth Traces

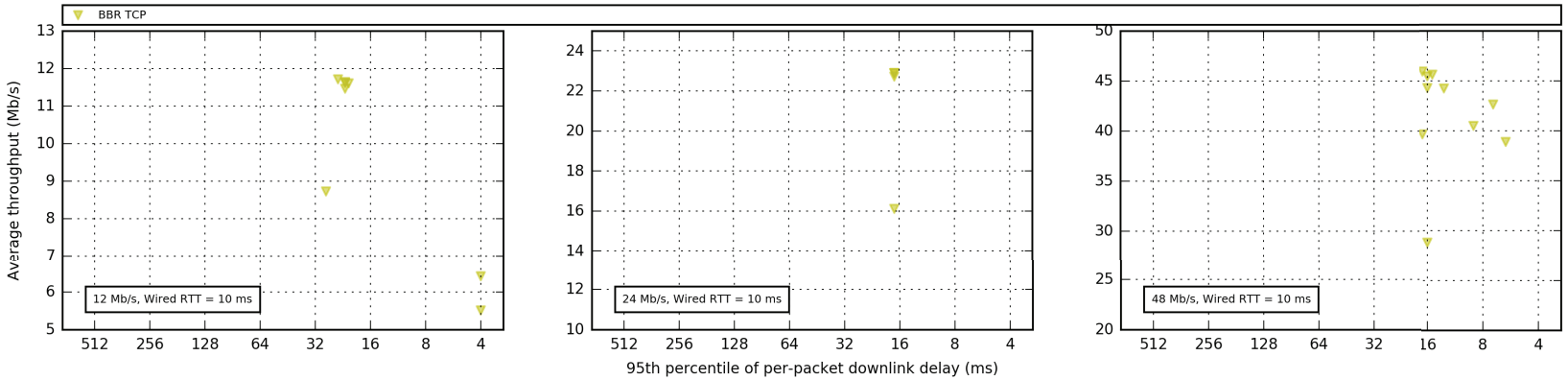


Performance With Fixed Bandwidth Traces



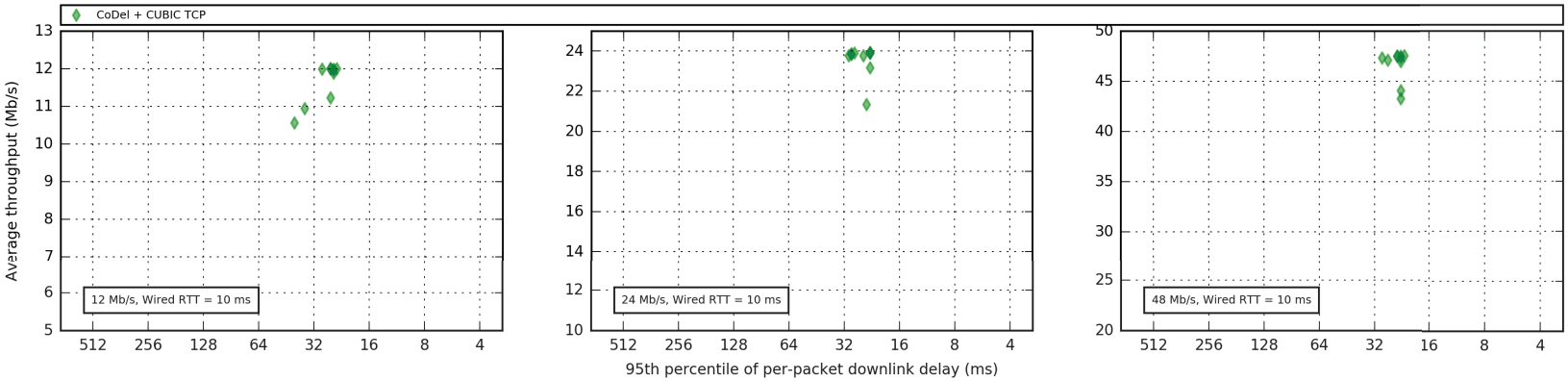
- LCTCP achieves consistent results across runs
- LCTCP does not incur unnecessary delay

Performance With Fixed Bandwidth Traces



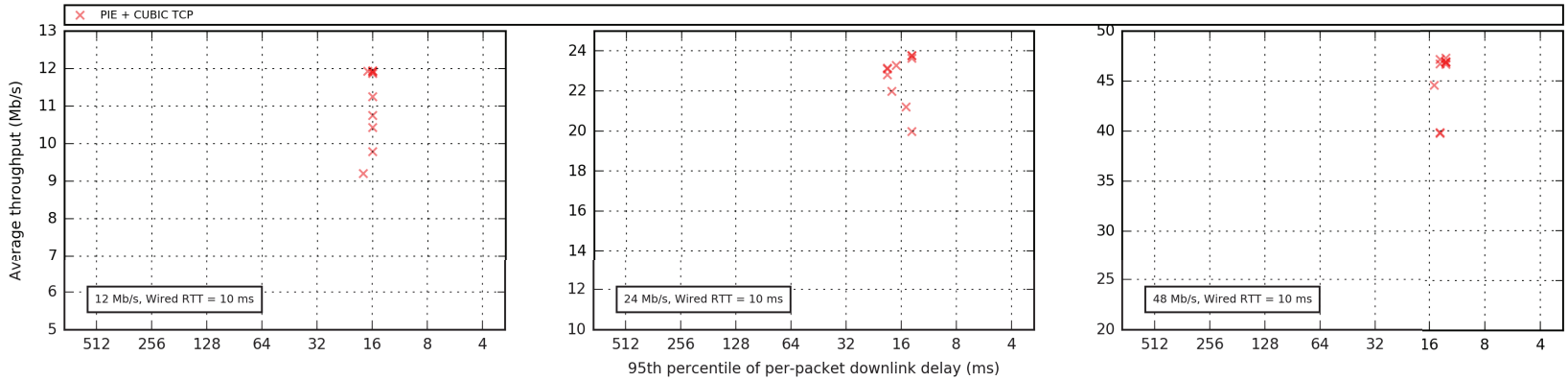
- LCTCP achieves consistent results across runs
- LCTCP does not incur unnecessary delay

Performance With Fixed Bandwidth Traces



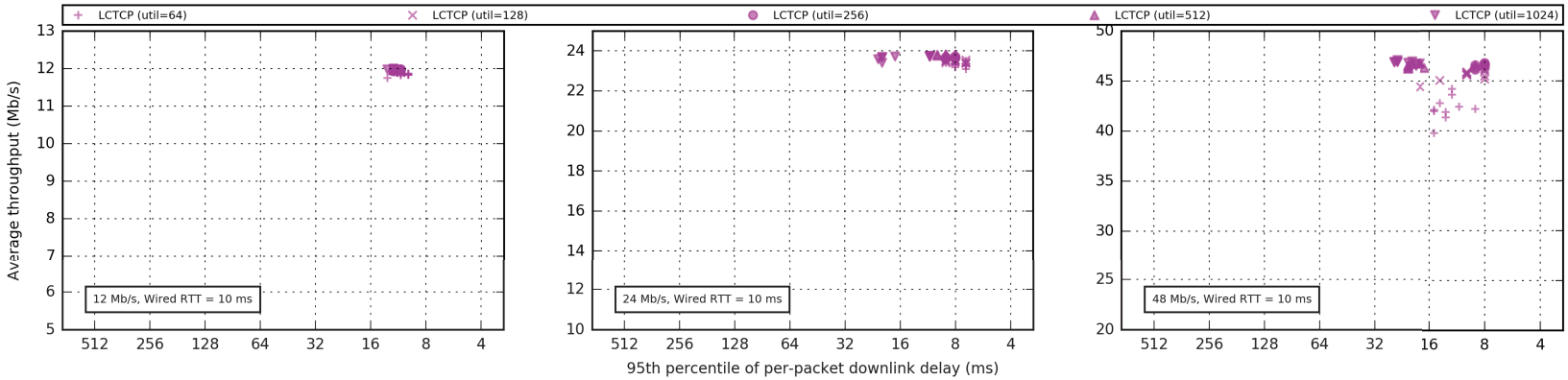
- LCTCP achieves consistent results across runs
- LCTCP does not incur unnecessary delay

Performance With Fixed Bandwidth Traces



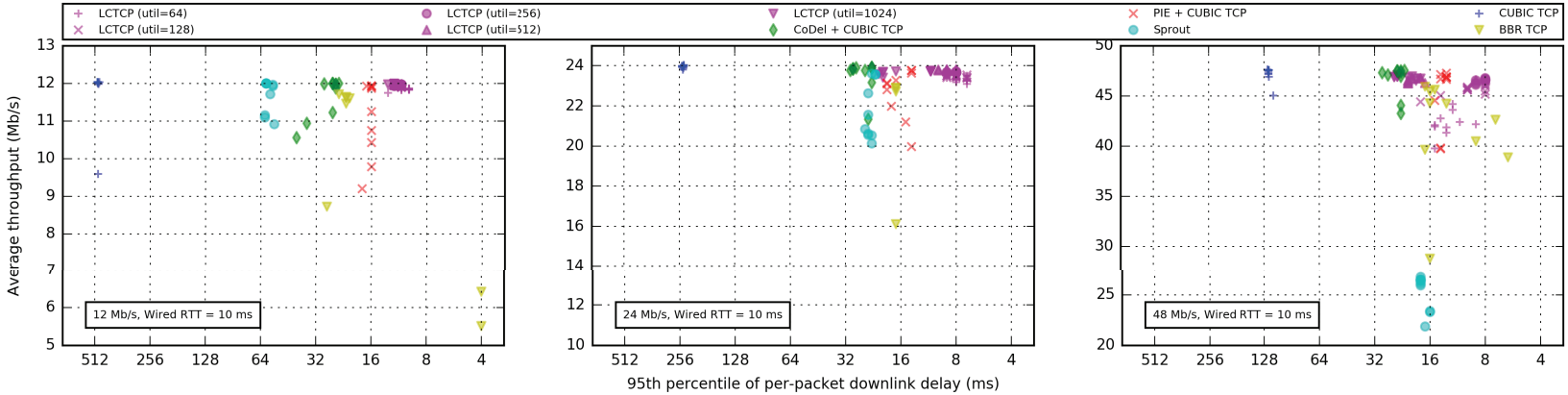
- LCTCP achieves consistent results across runs
- LCTCP does not incur unnecessary delay

Performance With Fixed Bandwidth Traces



- LCTCP achieves consistent results across runs
- LCTCP does not incur unnecessary delay

Performance With Fixed Bandwidth Traces



- LCTCP achieves consistent results across runs
- LCTCP does not incur unnecessary delay

Summary

Link-Coupled TCP:

- Leverages emerging trends in cellular networks and application architectures
- Allows network and application providers to cooperate and optimize for per-flow goals
- Provides accurate per-flow queue information to the server
- Enables each application to set its individual trade-off point between utilization and delay
- Is transparent to the client thus makes no irreversible changes to the architecture