ANRW 2021 – ACPF

Cheapest Path First Scheduling in Transport Layer Multi-Path Tunneling

COMPUTER SCIENCE
DATAVETENSKAP

Outline

Outline of **presentation**:

1) Introduction to ATSSS
2) Cheapest path first (CPF) scheduler & issues
3) Our solution (ACPF)

Outline

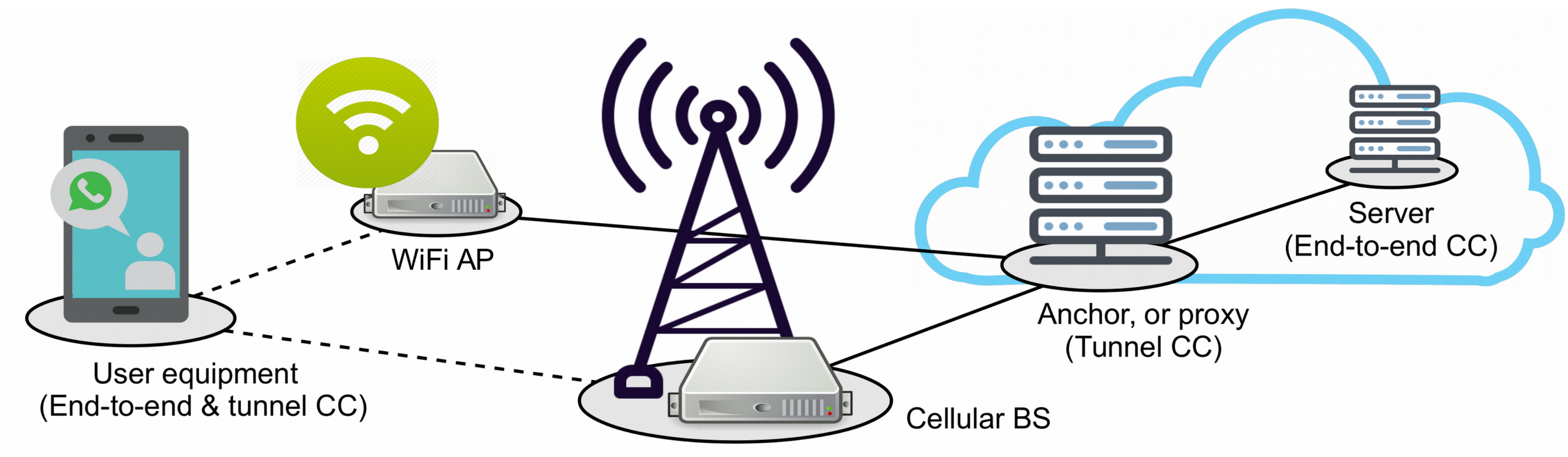
Outline of **presentation**:

1) **Introduction to ATSSS**

Introduction to ATSSS

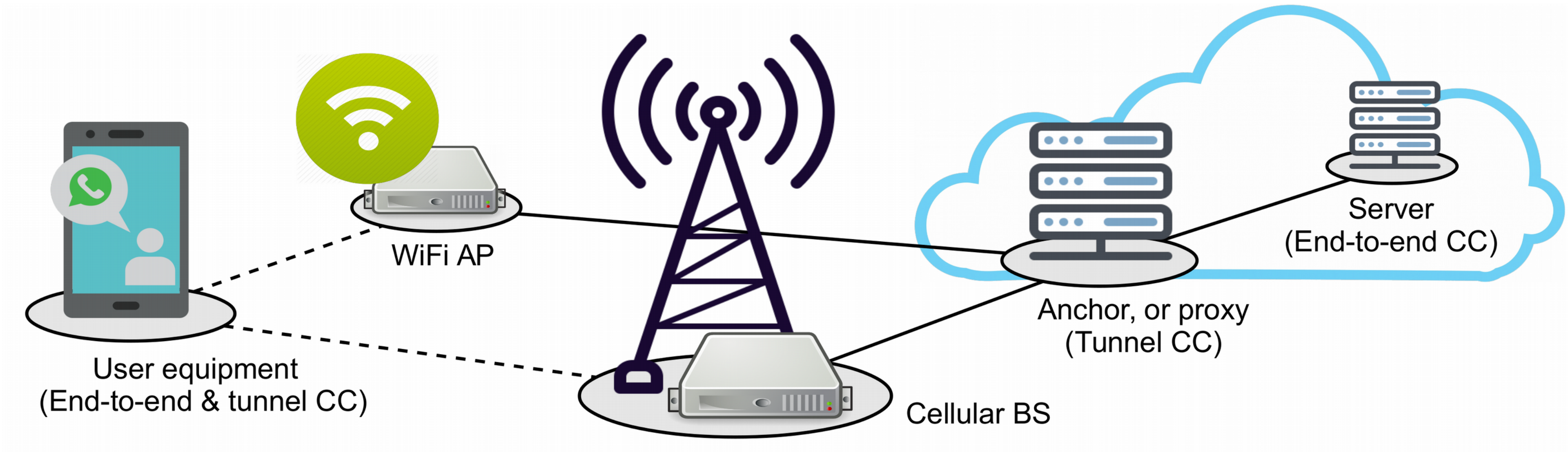
Access Traffic Steering, Switching and **Splitting**

- Transparent multipath between user and proxy
- TCP – MPTCP split approach for E2E TCP
- **Multi-path transpor-layer tunnel** for all other traffic



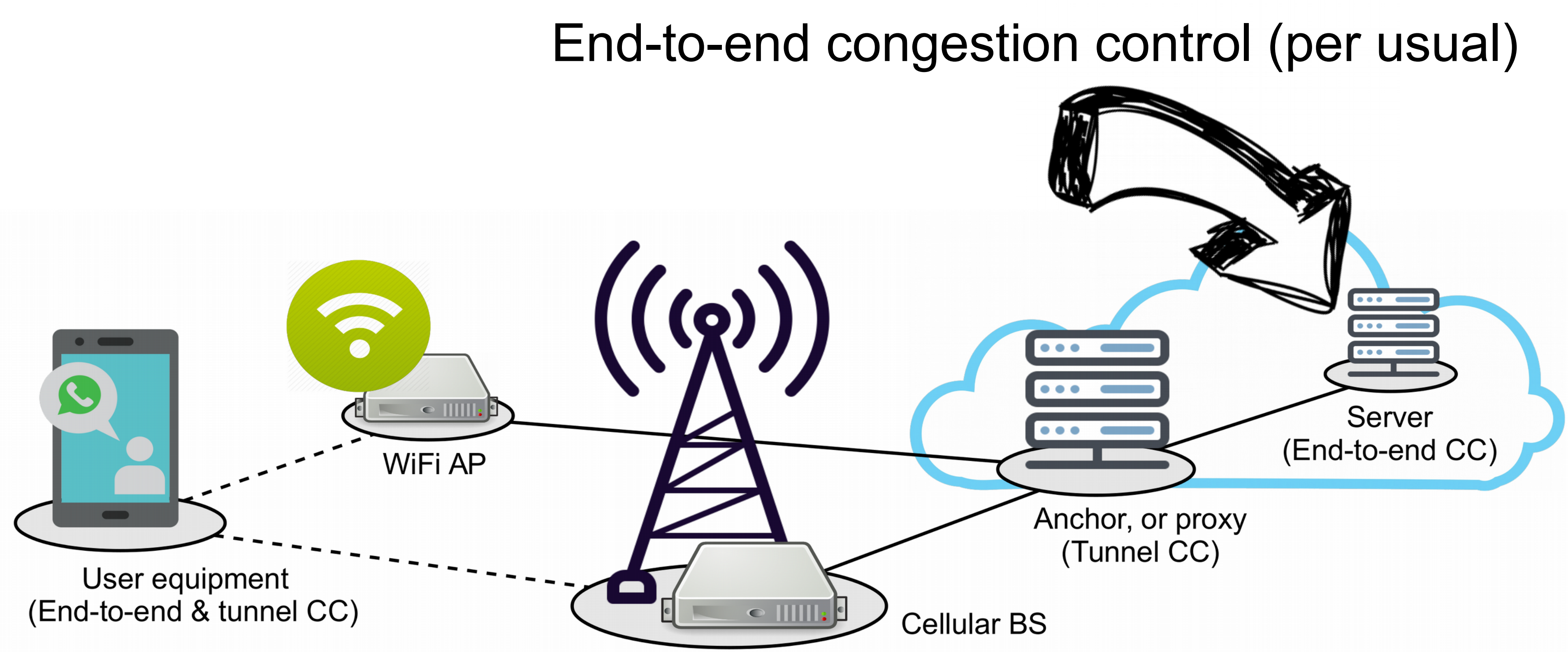
Introduction to ATSSS

Assuming download ...



Introduction to ATSSS

Assuming download …

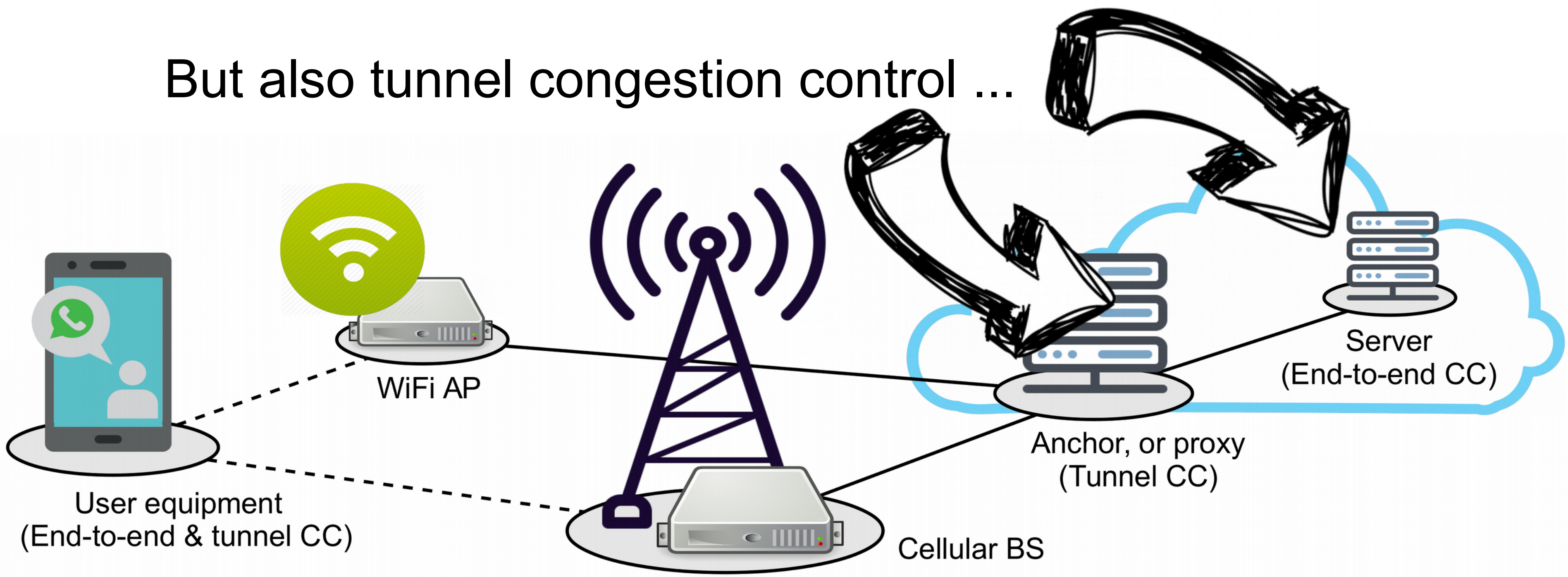


Introduction to ATSSS

Assuming download …

End-to-end congestion control (per usual)

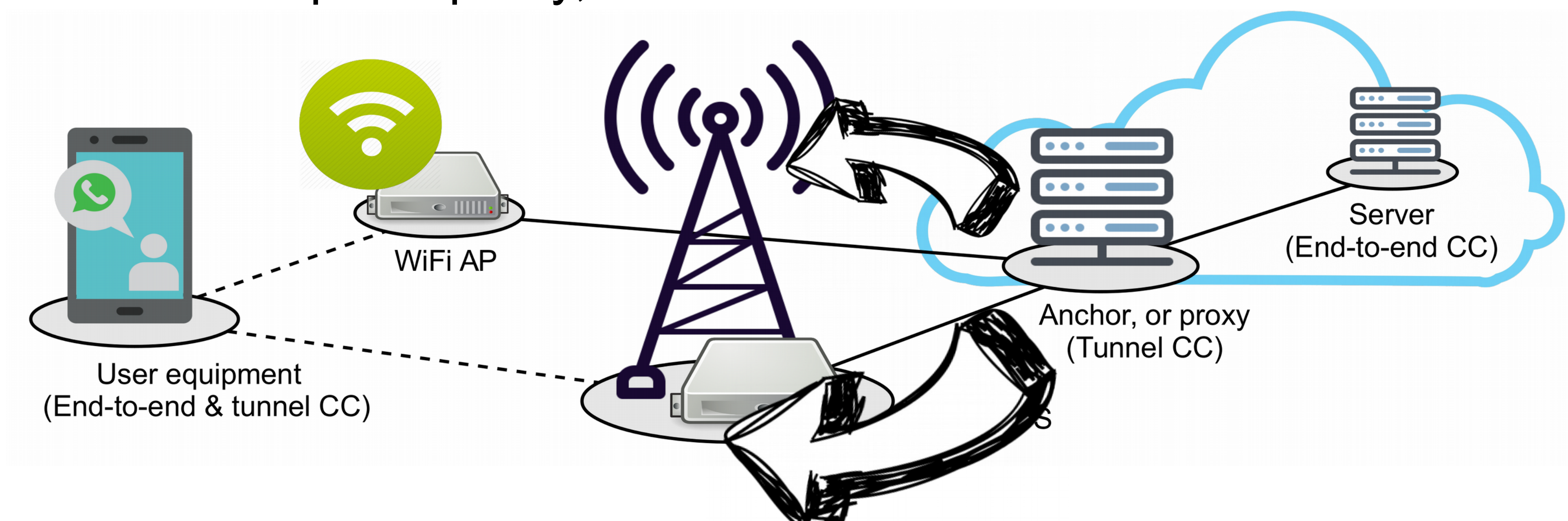
But also tunnel congestion control …



Introduction to ATSSS

Assuming download ...

Traffic is split at proxy, based on tunnel CC state ...

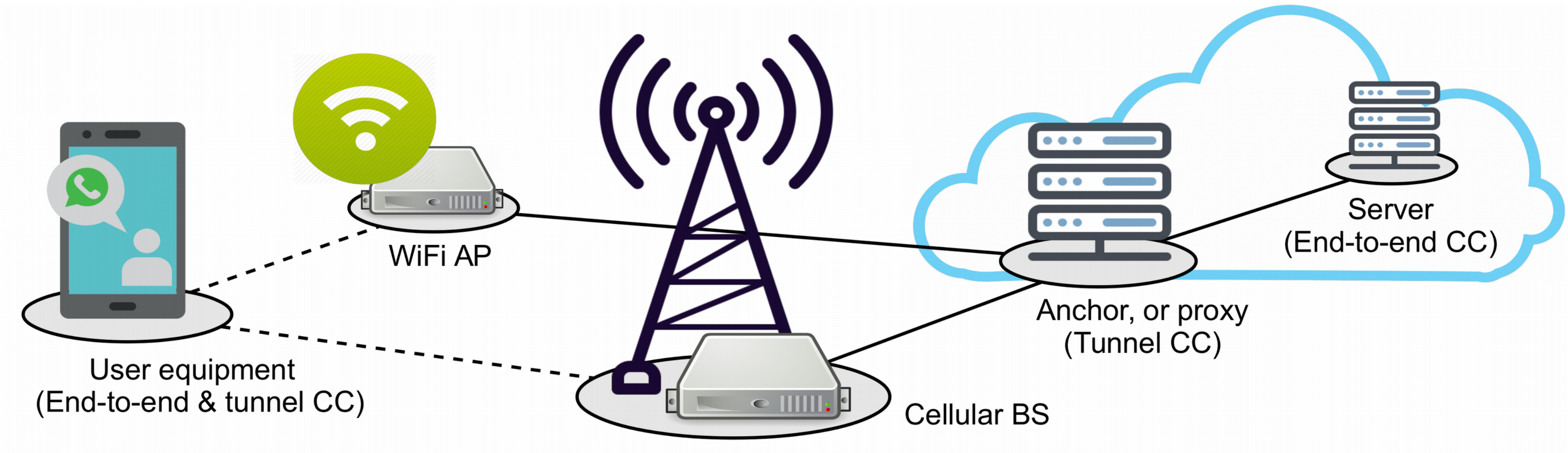


Introduction to ATSSS

Seem straightforward …

However, **issues** arise from interaction between:

- End-to-end congestion control
- Tunnel congestion control
- Tunnel scheduler



Outline

Outline of **presentation**:

1) Introduction to ATSSS
2) **Cheapest path first (CPF) scheduler & issues**

2.1 CPF

Cheapest Path First

WiFi is cheap, cellular less so

Using cheapest path first = attractive

→ **Cheapest path first** scheduler

Rank paths by cost in ascending order, then:

```
1)  get packet
2)  index = 1
3)  while index < number of paths:
4)      if cwnd[index] > in-flight[index]:
5)          send packet over path[index]
6)          goto 1
7)      index++
8)  wait until state change and goto 2
```

Cheapest Path First

2.1 CPF

Problematic, **bad definition of “first”** …

```
1)  get packet
2)  index = 1
3)  while index < number of paths:
4)      if cwnd[index] > in-flight[index]:
5)          send packet over path[index]
6)          goto 1
7)      index++
8)  wait until state change and goto 2
```

Cheapest Path First

2.1 CPF

When condition holds

→ path is often **fully utilized** (which is good)

→ but also often **congested** (which is bad)

```
1)  get packet
2)  index = 1
3)  while index < number of paths:
4)          if cwnd[index] > in-flight[index]:
5)                  send packet over path[index]
6)                  goto 1
7)          index++
8)  wait until state change and goto 2
```

Cheapest Path First

Note: Issue is not congestion *in itself*

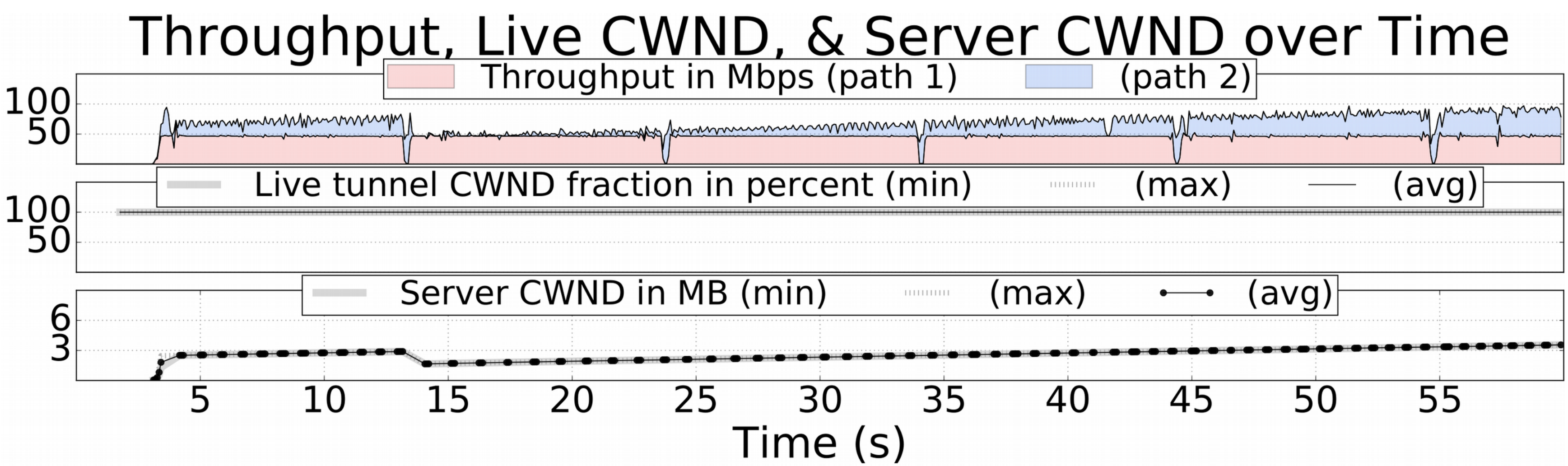
→ congestion also *prevents aggregation*

1) get packet
2) index = 1
3) while index < number of paths:
4)   **if cwnd[index] > in-flight[index]:**
5)   send packet over path[index]
6)   goto 1
7)   index++
8) wait until state change and goto 2

CPF Issue

Example result:

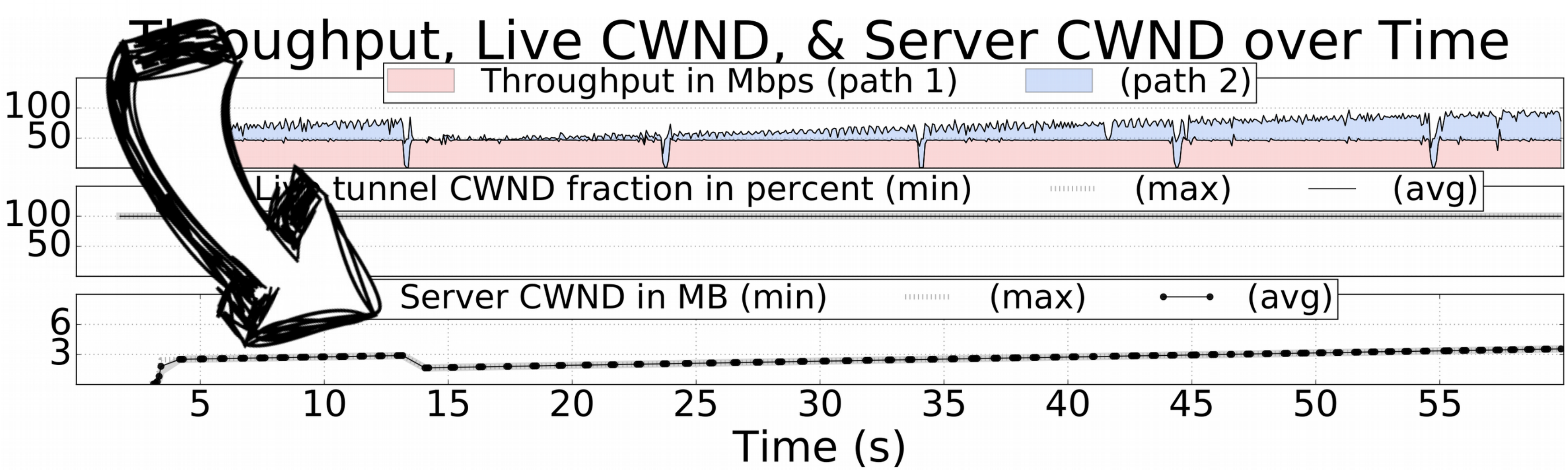
- Setup in Mininet
- Greedy TCP flow
- TCP-NewReno over TCP-BBR
- 50 + 50 Mbps paths, 32 + 52 ms RTT
- FIFO, size > typical BBR CWND



CPF Issue

Example result:

Reduction in server CWND ...

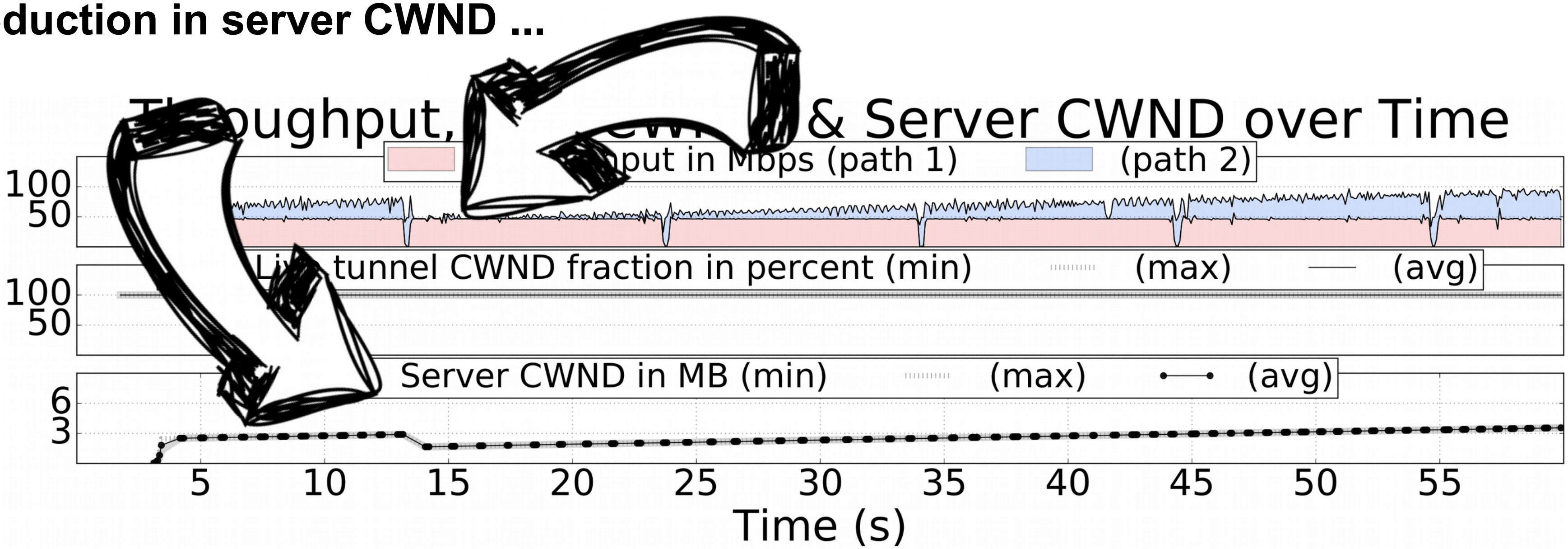


CPF Issue

Example result:

… reduction in throughput

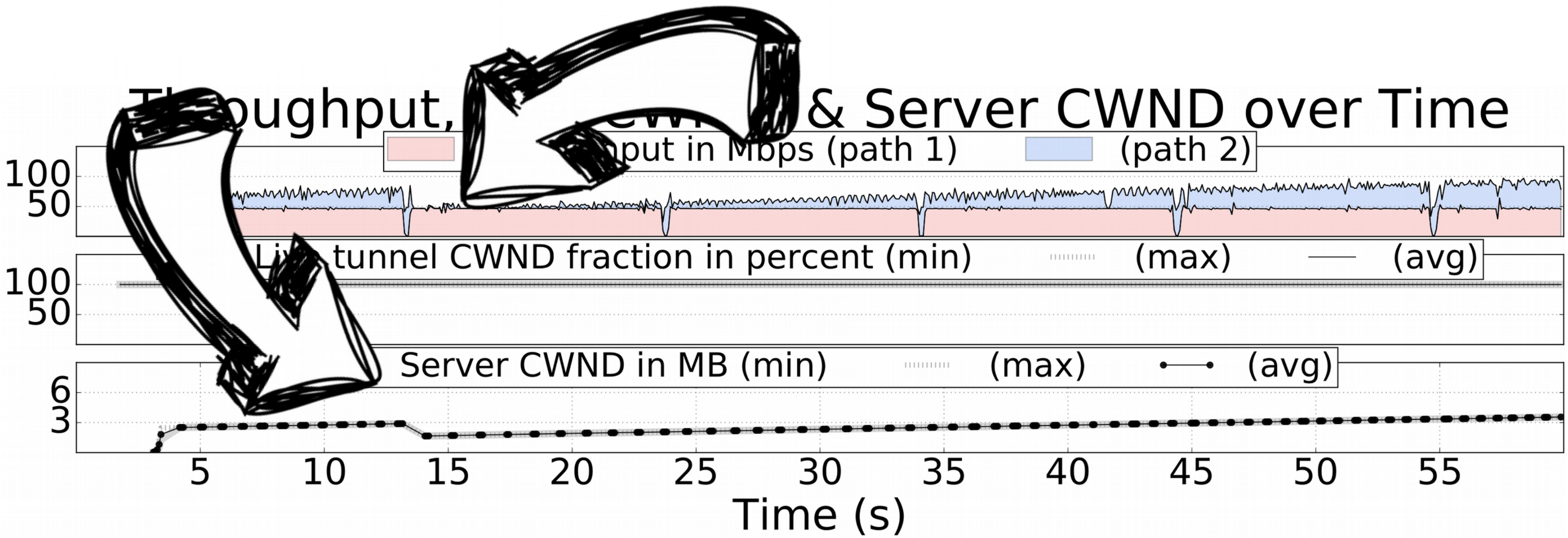
Reduction in server CWND …



CPF Issue

Summary of state after event at arrows:

- Primary path is very likely (somewhat) **congested**
- Yet, multipath capacity is clearly **under utilized**
  → These two *should not happen simultaneously*
- Think of this as: Bad distribution of server CWND



Outline

Outline of **presentation**:

1) Introduction to ATSSS
2) Cheapest path first (CPF) scheduler & issues
3) **Our solution (ACPF)**

Adaptive CPF

Basic idea of ACPF:

- Distribute server CWND to *maximize throughput*
  (Happy side effect: will often also reduce congestion)

How its done:

- Determine bandwidth-(minimum-)delay product (BDP)
- Add *live CWND* concept: Fraction of CWND allowed for scheduling
- BDP <= live CWND <= 100 % of CWND
  (In our case, tunnel with BBR, assume lower limit is half of CWND)

Manage live CWND by periodically:

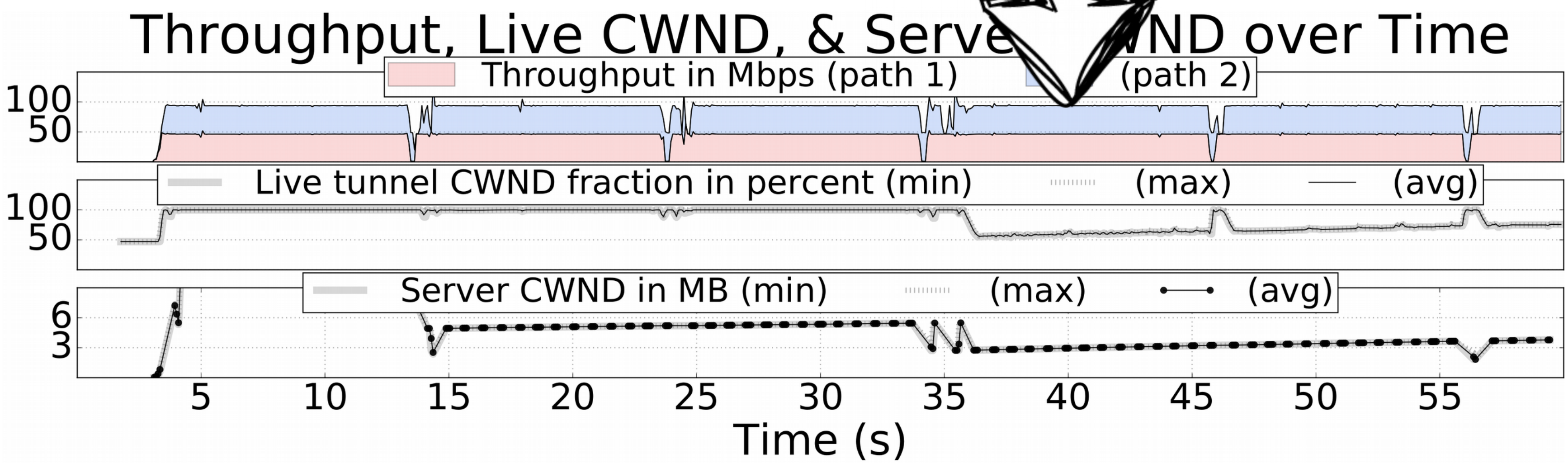
1) Increasing it, if there is a consistent sched. queue occupancy
2) Decrease otherwise

ACPF Performance

Example result, same config as before, with ACPF:

Always full utilization …

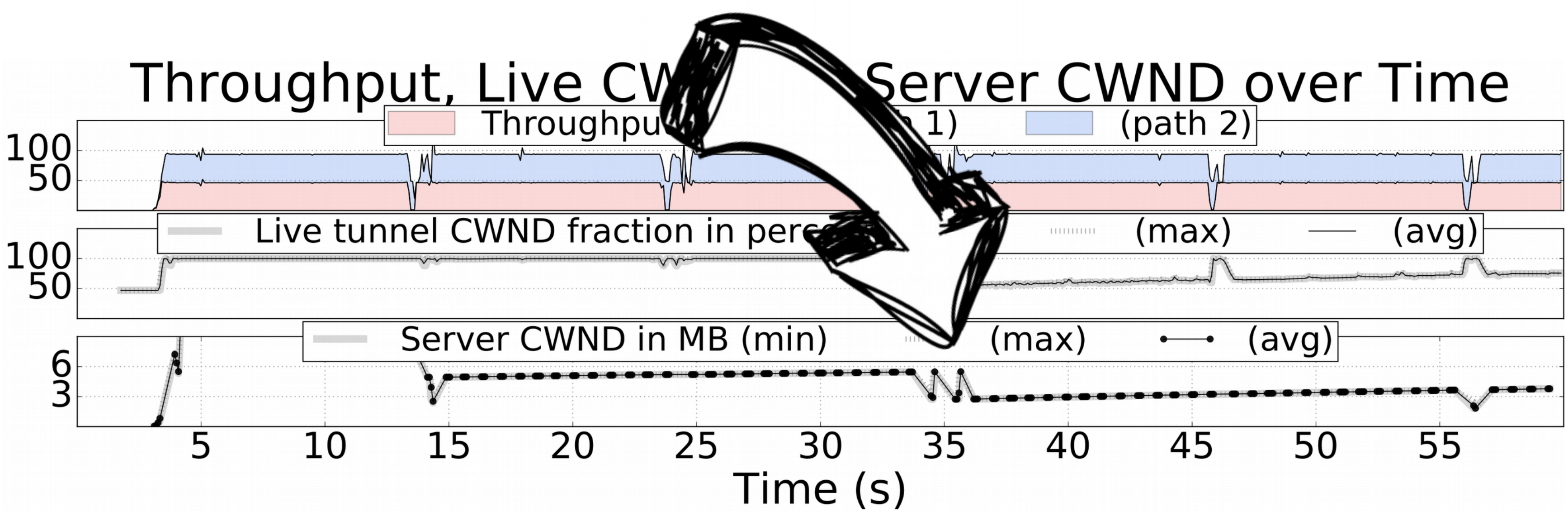
I.e, solution works!



ACPF Performance

Example result, same config as before, with ACPF:

Server CWND reduction like before, why still full utilzation?

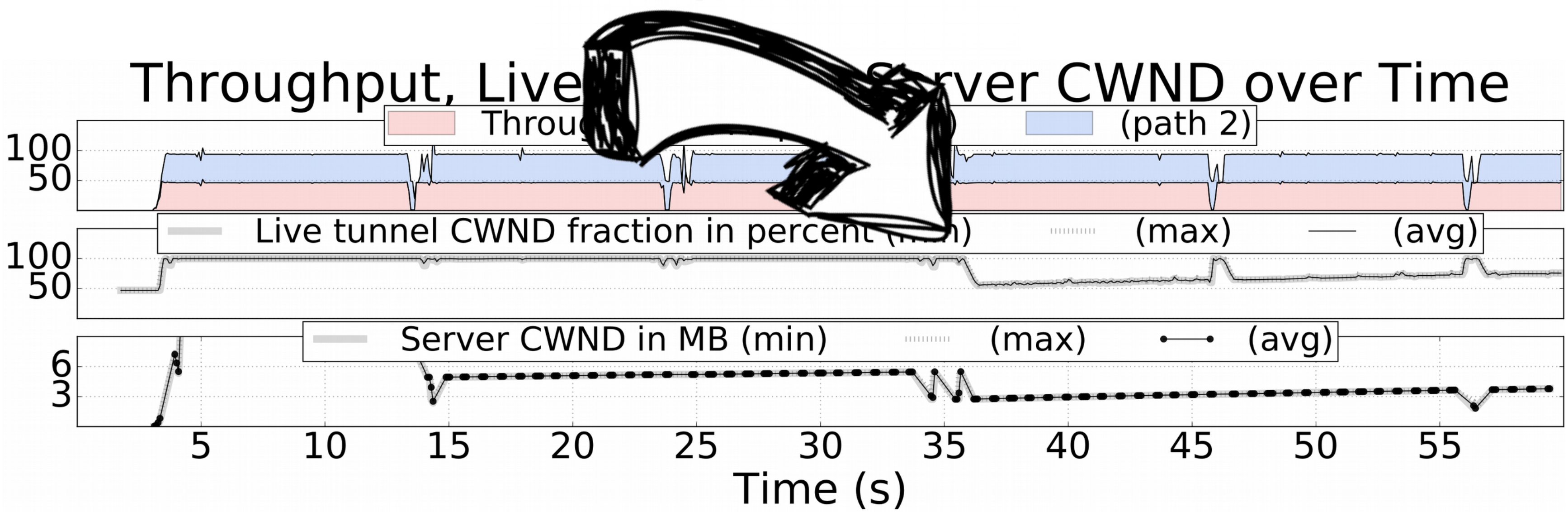


ACPF Performance

Example result, same config as before, with ACPF:

Server CWND reduction like before, why still full utilzation?

Answer is: ACPF compensate with live CWND reduction



ACPF Conclusion

Example above showed:

1) ACPF maintained full utilization
2) It reduced live CWND when it had to
   (Good guess: Bad utilization, had live CWND been at 100%)

More detailed evaluation in paper

Future improvements:

- Better integration with BBR to get actual BDP (**is critical**)
- Make sure BDP is accurate when throughput is very dynamic
- Evaluate performance for soft handover (WiFi → LTE) scenario
  (Switching in ATSSS, as opposed to splitting)

ANRW 2021 – ACPF

Cheapest Path First Scheduling in Transport Layer Multi-Path Tunneling