Designing a Multipath Transport Protocol for Serval

Torin Rudeen
Princeton University
SNS Group Meeting
18-Feb-2013

Serval

A Leap Toward Next-Generation Service-Centric Networking

Multipath TCP (MPTCP)

- Functioning multipath version of TCP
- Open source implementation for Linux
- Undergoing standardization process

MPTCP vs. Serval

MPTCP

- “How can we make multipath deployable in today's internet?”
- Incremental
- Multipath only
- Achieve backwards compatibility via additional complexity

Serval

- “How can we reimagine the network stack for today's internet?”
- Transformative
- Multipath, migration, service names, etc.
- Find simplest design without worrying about backwards compatibility.

MPTCP and Serval have different (but partially overlapping) design goals and contexts. Multipath for Serval should borrow from MPTCP where applicable, but find different solutions where requirements differ.

Anatomy of TCP

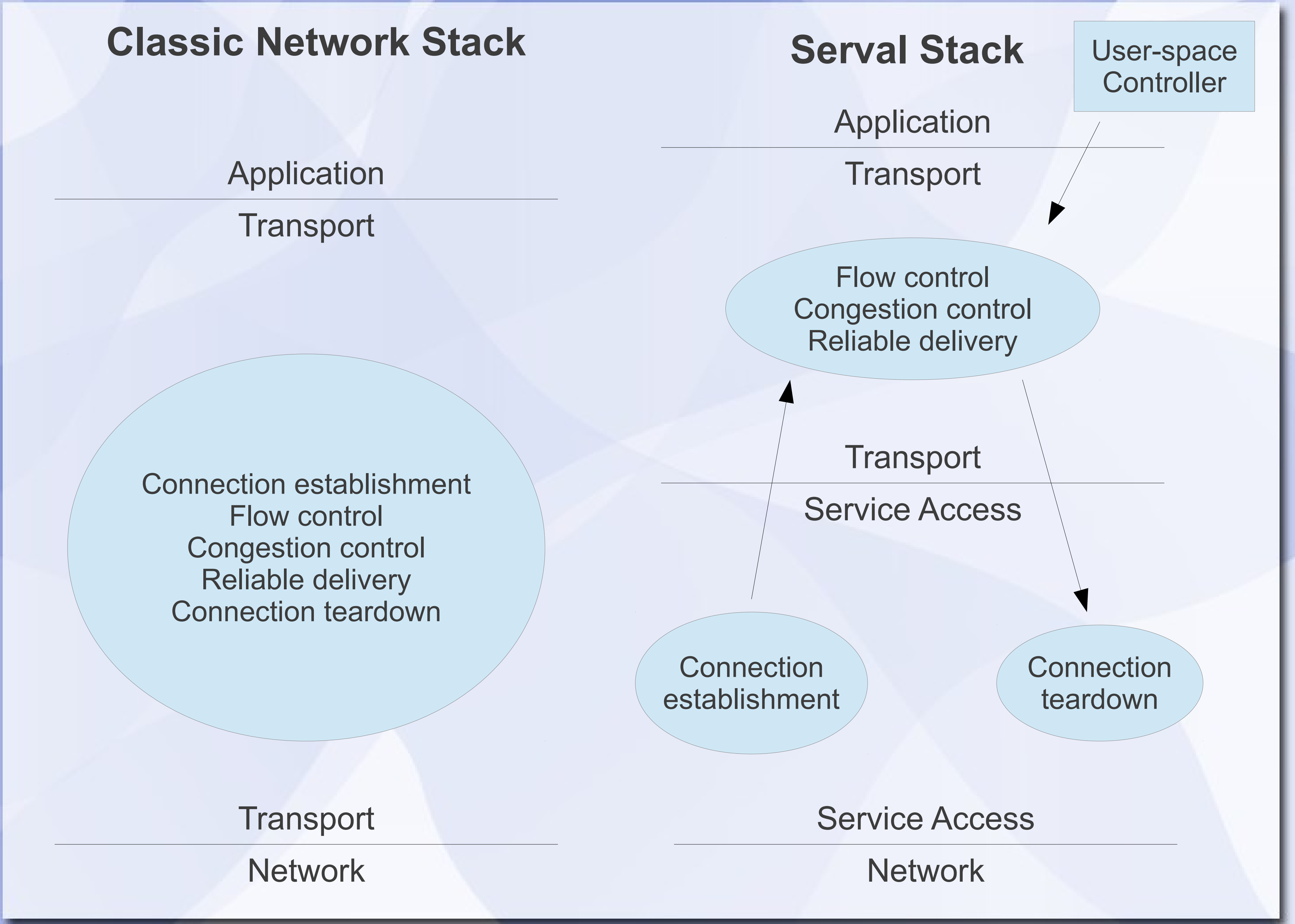
TCP does five things: [1]

- Connection establishment
- Flow control
- Congestion control
- Acknowledgements and reliable delivery
- Connection teardown

Anatomy of TCP

TCP does five things: [1]

- Connection establishment
- Flow control
- Congestion control
- Acknowledgements and reliable delivery
- Connection teardown


Classic Network Stack
Application
Transport
Connection establishment
Flow control
Congestion control
Reliable delivery
Connection teardown
Transport
Network
Serval Stack
User-space Controller
Application
Transport
Flow control
Congestion control
Reliable delivery
Transport
Service Access
Connection establishment
Connection teardown
Service Access
Network

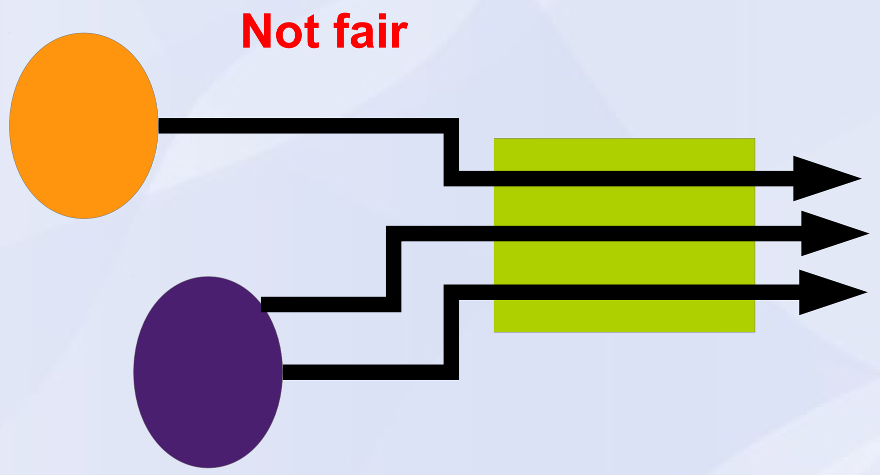
Connection Establishment/Teardown

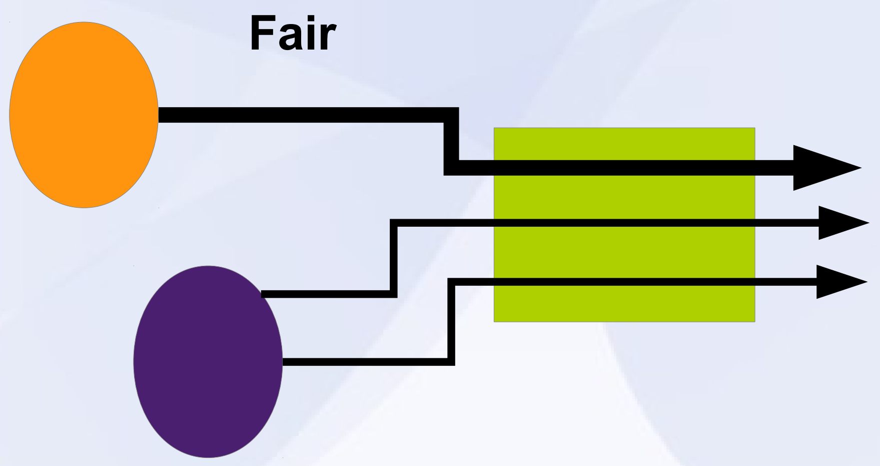
- Handled in the Service Access Layer
- Implemented this summer
- See Matvey's ECCP paper [2] for protocol details.
- Talk to me for implementation details.
- See “mptcp” git branch.

Congestion Control

- Multipath congestion control algorithm should improve throughput for the end hosts, while satisfying fairness constraints.
- In other words, make things better for me without making them worse for everyone else.
- Researchers working on MPTCP congestion control have translated this principle into three goals for any multipath congestion control algorithm: [4]
  - Do no harm
  - Improve throughput
  - Balance congestion

Do No Harm





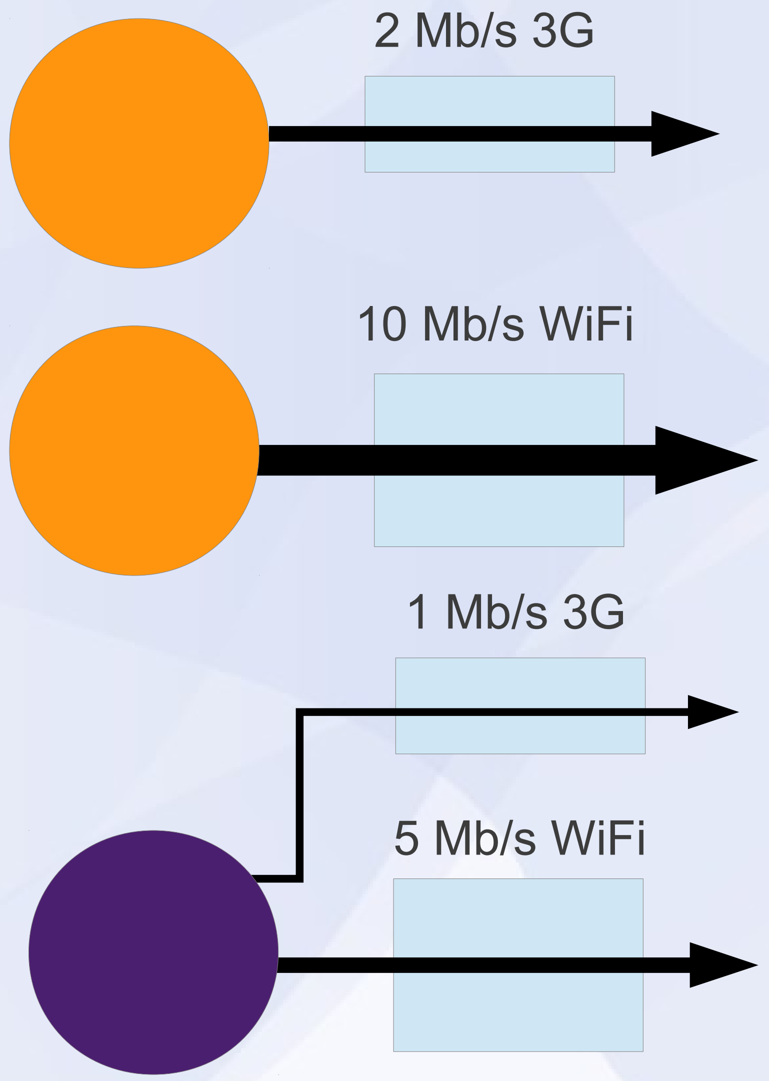
A multipath connection should not use up more bandwidth at a shared bottleneck than a single path connection.

Strawman #1:
Run single path TCP on each flow independently.

If it weren't for this criterion, we would trigger a flow “arms race,” where whichever host created the most flows would control the largest share of the resource.

Example from [3]

Improve Throughput

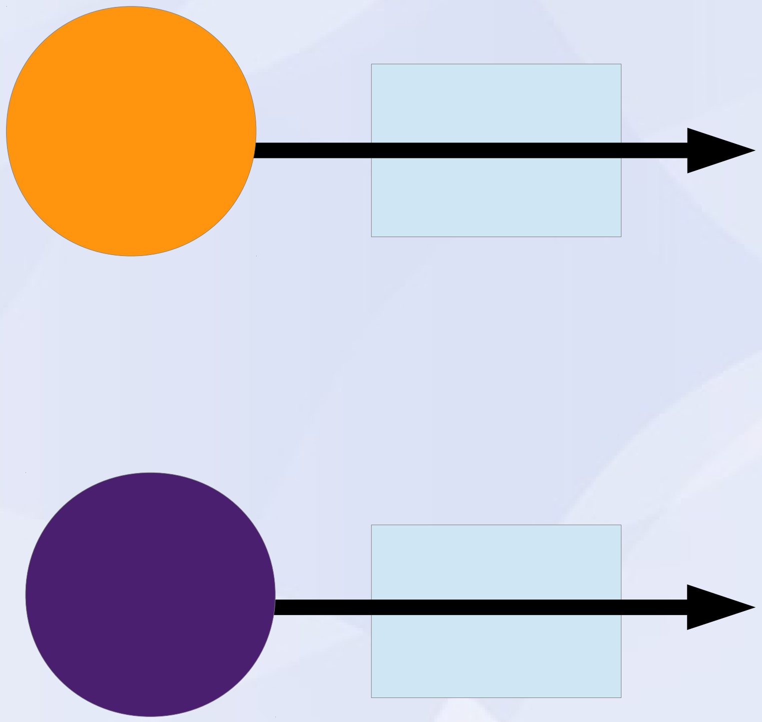


A multipath connection should provide at least as much aggregate bandwidth as a single path connection on the best of the available links.

Strawman #2:
Run single path TCP on each flow independently, scaling aggressiveness by 1/n, where n is the number of flows.

Example from [3]

Balance Congestion

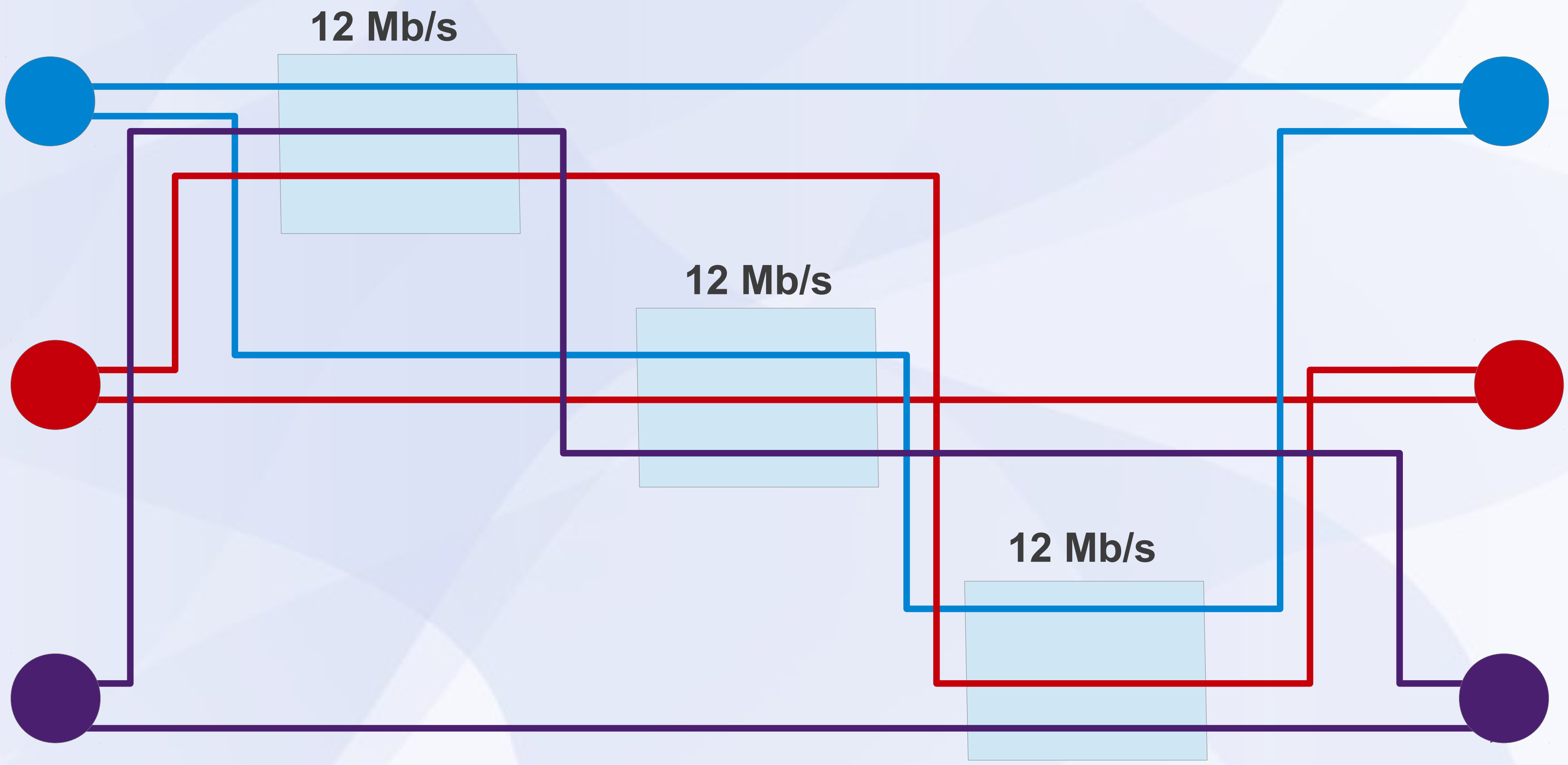


Strawman #3:
Run single path TCP.

“A multipath flow should move as much traffic as possible off its most-congested paths, subject to meeting the first two goals.” [4]

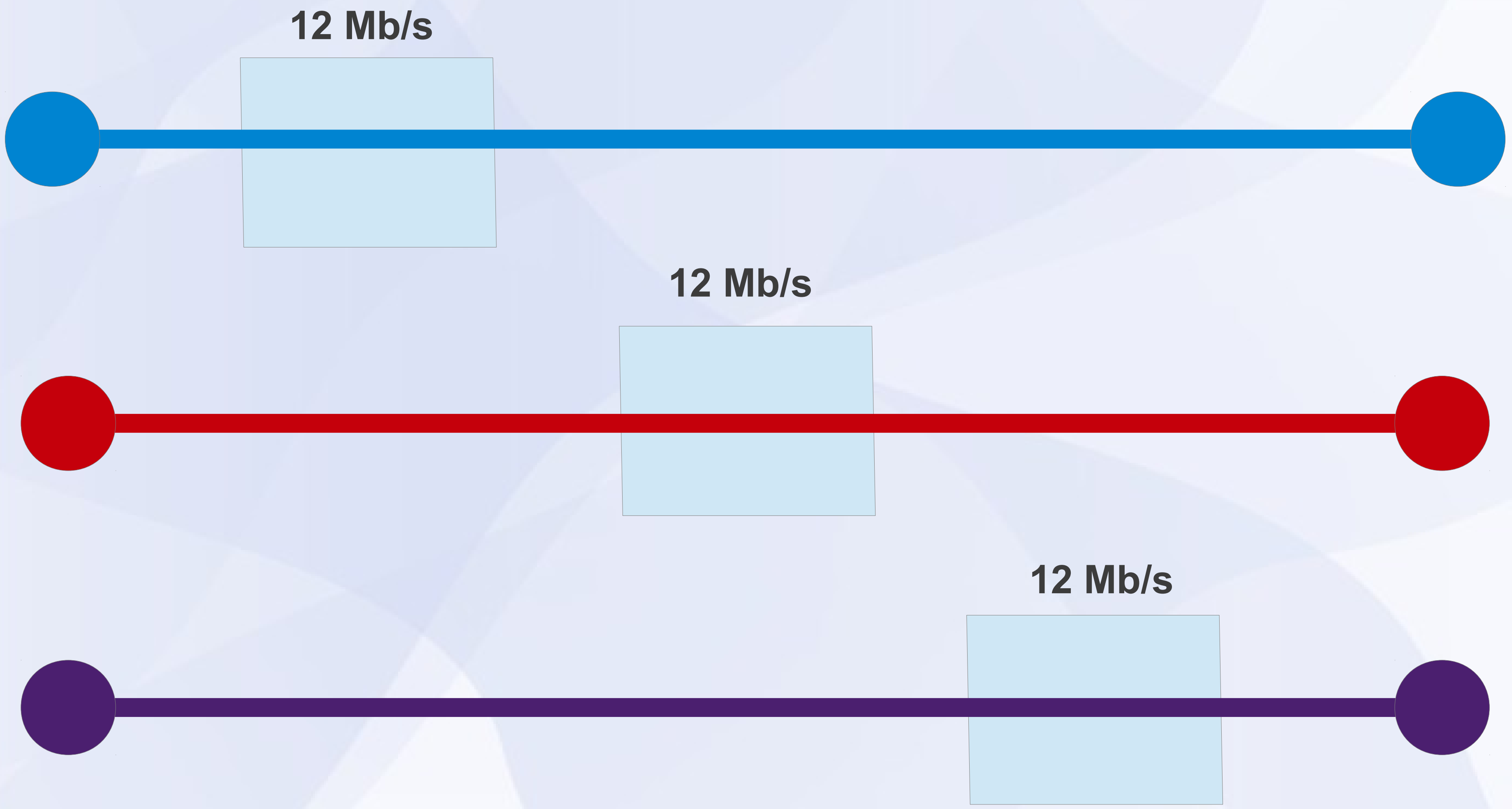
Multipath must provide a benefit over that provided by single path.

Balance Congestion



Example from [3]

Balance Congestion



Example from [3]

The MPTCP Congestion Control Algorithm

As described in [3]:

- For each ACK on a subflow r, increase w_r, the subflow's window size, by $\min(alpha/w\_total, 1/w\_r)$.
- For each loss on a subflow r, decrease w_r by $w\_r/2$.

$alpha$ is an overall aggressiveness parameter, selected so as to satisfy “do no harm” and “improve throughput” conditions.

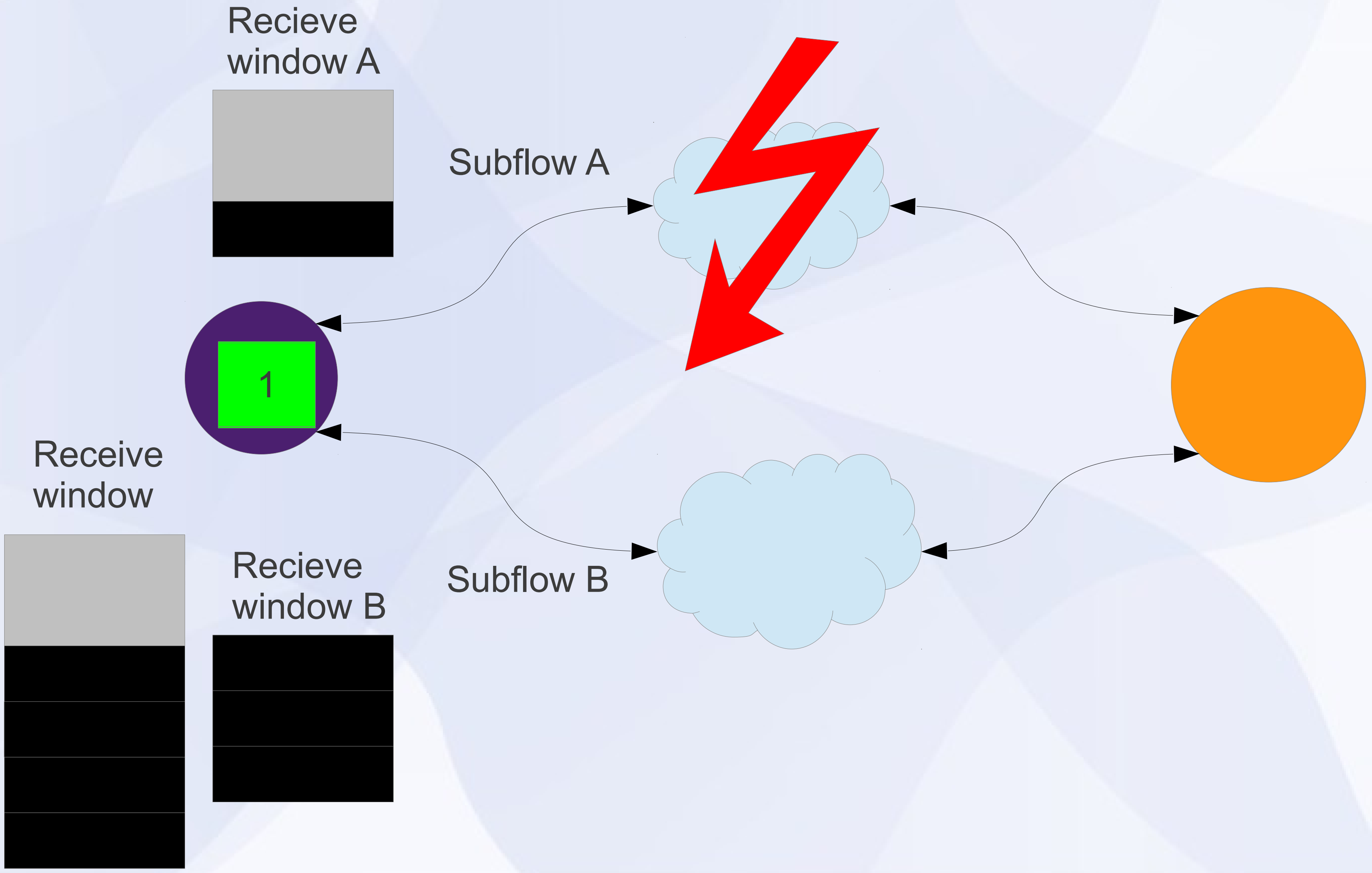
This moves most traffic to less congested flows, but keeps probing unused pathways to adapt to changing network conditions.

See [3] for algorithm derivation and proof, [8] for implementation suggestions and specifications.

Flow Control

- Per subflow, or per connection?
- Congestion control is per subflow
- Per subflow simpler
- Unfortunately, per subflow flow control leads to deadlock!

Per subflow flow control allows deadlock [1]



Path State vs Connection State

To recap, we have established:

- Congestion control must be managed per path
- Flow control must be managed per connection

How to implement?

- MPTCP uses separate sequence numbers for subflows and connections, along with separate ACKs.
- Overcomplicated, mainly to get by middleboxes.
- Can we do better?

Reliable Delivery and Acknowledgements

- Reordering, retransmission, ACKs, and more.
- We have to decide on a sequence number scheme.
  - Per connection sequence nums?
  - Per subflow and per connection sequence nums?
- This is a tradeoff between connection-level complexity and subflow-level complexity.
- Choice will influence implementation of flow control and congestion control.

MPTCP's Solution

- Separate sequence spaces—per subflow and per connection
- Use TCP sequence nums for per subflow sequence nums
- Use TCP options for per connection sequence nums
- Send special connection-level ACKs using TCP options
- Mainly motivated by backwards compatability concerns:
  - Middleboxes don't like discontinuous sequence nums
  - “Disguise” subflow as a single path TCP connection

Our Solution

?

Dumb Subflows, Smart Connection

As compared to MPTCP's “Smart Subflows, Smart Connection”

- Manage everything at the connection level. One connection-level sequence number space.
- Subflows have little state.
- Everytime a packet needs to be sent, just pick a subflow.
- ACKs and retransmitted packets don't need to be on the same subflow as the original packet.
- Flow control and acknowledgements become very simple

Issue: Congestion Control

- If we use a single sequence number space, how can we implement congestion control on a per subflow basis?

Approach #1: SACK accounting

- Remember which packets we sent over which interface, use SACKs to deduce which have been received and which are in flight.
- Possibly complicated.

Approach #2: Approximate congestion control

- Remember how many packets have been sent over the subflow recently.
- Use estimated RTT to estimate how many packets are in flight.
- Use this value instead of a congestion window.
- Still need to calculate RTT and bandwidth.

Issue: Some Subflow State Needed

- For congestion control, we need to know:
  - When a packet is lost, which subflow was it sent on?
  - When an ACK is received, which subflow was the packet sent on?
- MPTCP keeps entire TCP state for each subflow.
- Is there some sort of lightweight state we can keep that meets these needs?

Issue: Excessive Reordering

Varying RTTs mean massive reordering, requiring large buffers and lots of CPU time to correct.

Optimization 1: Allocate packets to interfaces in large chunks.

- Potential issues if one interface is much slower or fails before it sends its entire chunk.
- Requires per subflow state to assemble the chunks on the receiver end.

Optimization 2: Predictive transmission

- Don't send packets in order, send so that they'll *arrive* in order, based on estimated delay-bandwidth product.

Other Issues

- Userspace controller—controls what?
  - Subflow creation/termination?
  - Traffic allocation between subflows?
  - How?

Thank you!

References

1. Raiciu et al. "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP"
2. Arye, Matvey. "A Formally-Verified Migration Protocol For Mobile, Multi-Homed Hosts"
3. Wischik et al. "Design, Implementation, and evaluation of congestion control for multipath TCP"
4. Raiciu et al. "Practical Congestion Control for Multipath Transport Protocols"
5. Wischik et al. "Control of multipath TCP and optimiziation of multipath routing in the Internet"
6. Nordstom et al. "Serval: An End-Host Stack for Service-Centric Networking"
7. Podmayersky, Brandon. "An Incremental Deployment Strategy for Serval"
8. Raiciu et al. "Coupled Congestion Control for Multipath Transport Protocols" (RFC)