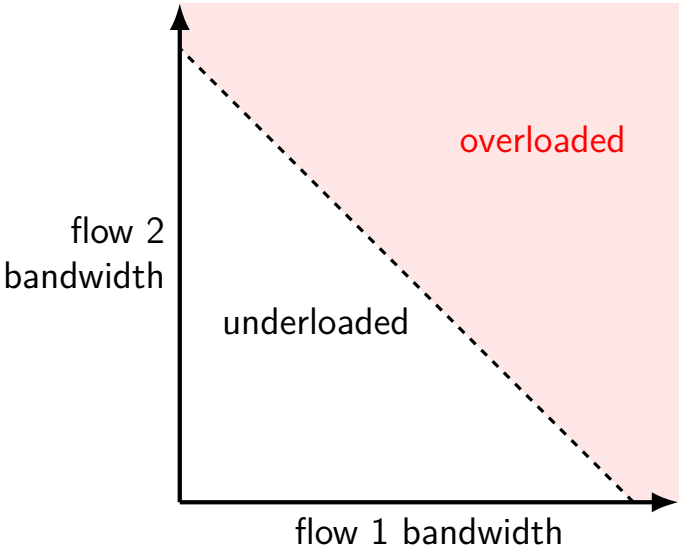
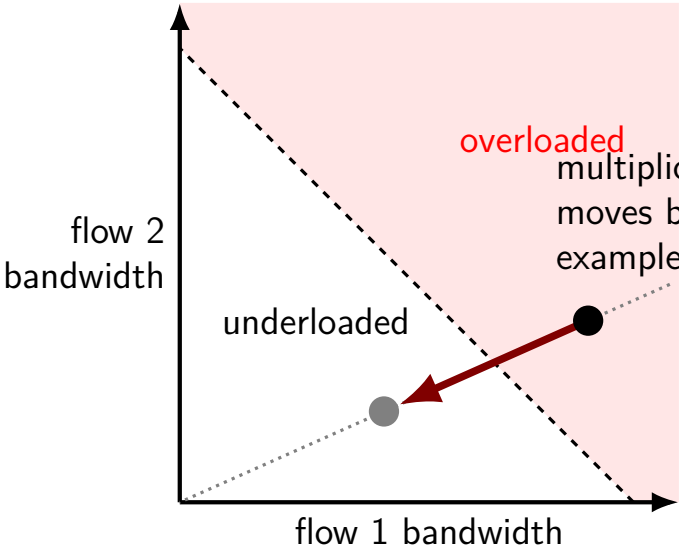
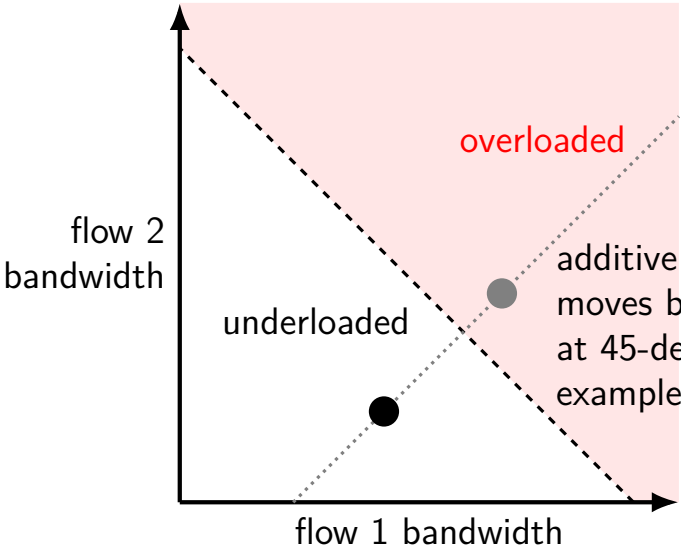
# picturing sharing



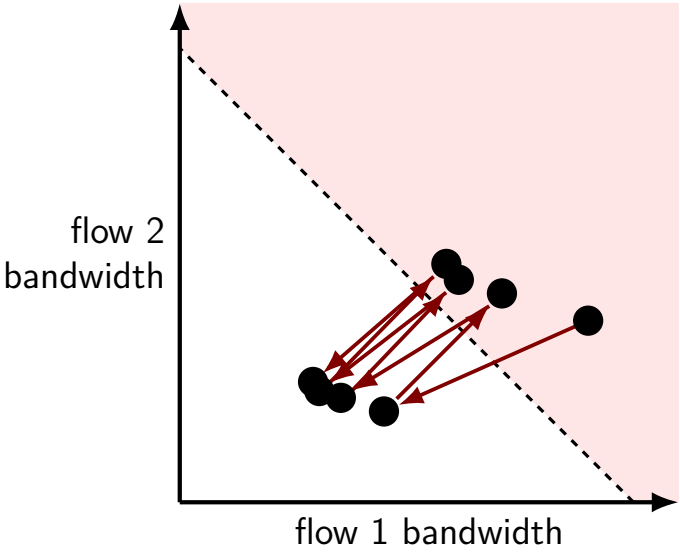
# picturing sharing



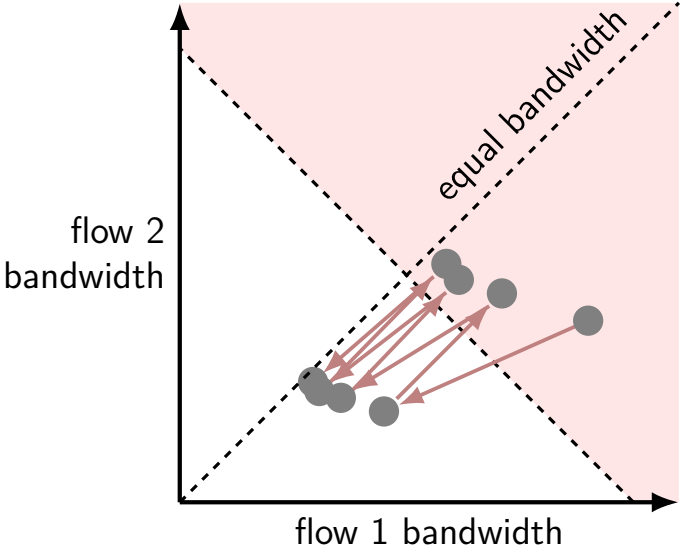
# picturing sharing



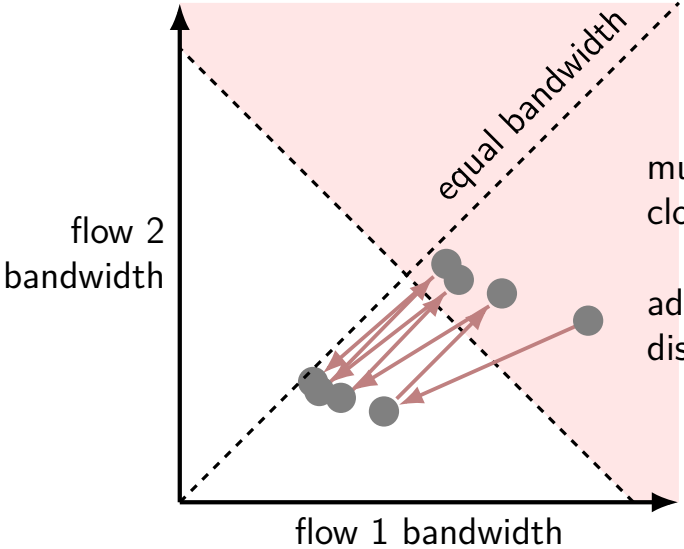
# picturing sharing



# picturing sharing



# picturing sharing




multiplicative decrease brings closer to equal bandwidth line

additive increase keeps same distance from line

# models that give numbers?

deciding on congestion control

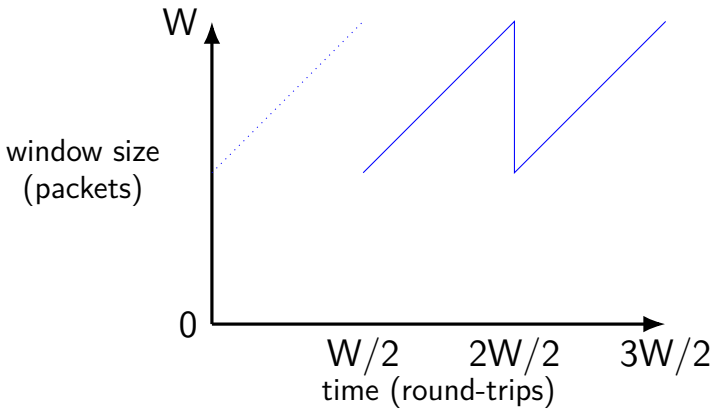
would like to do math to say how well they'll do

common approach: but gets complicated

simple example: estimating average rate of TCP-like AIMD with no congestion

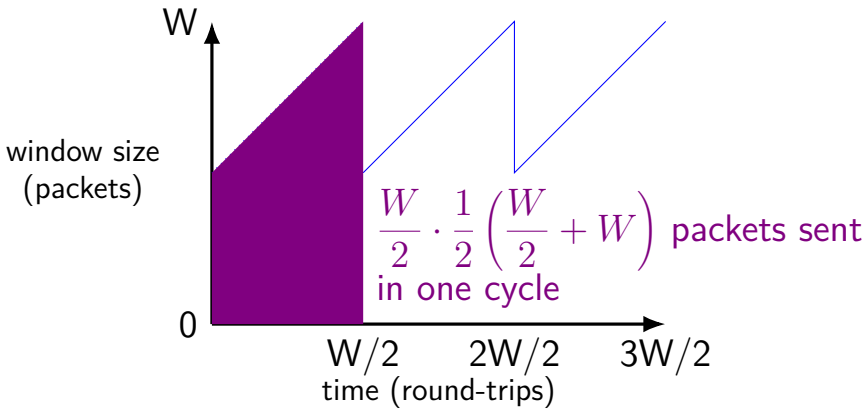
# models that give numbers? (1)

figure adapted from Mathis et al, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm"



# models that give numbers? (1)

figure adapted from Mathis et al, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm"



# models that give numbers? (2)

$\frac{W}{2} \cdot \frac{1}{2}\left(\frac{W}{2} + W\right)$ per $W/2$ round trips

$\frac{3W}{4}$ packets per round trip time

packet loss should occur when window size $=$ bandwidth-delay product

at the capacity of the link(s)

so $W =$ link BW $\cdot$ RTT

$\implies$ 3/4 link BW achieved total

# exercise: what things did this model miss?

# some answers

RTT/delay depends on how many packets queued

packet loss could occur for other reasons

- competing connections
- network errors

'bursty' connection could trigger packet loss earlier

extra packets being sent for retransmissions

packet loss could trigger timeout/multiple decreases

behavior of other connections sharing links

delays in sending ACKs depending how fast receiver's CPU is

...

# more sophisticated models?

we can add to formulas to account for other things

this is something people do, but...

most common technique is *discrete event simulation*

# interlude: loss rate → transfer rate

adapted from Mathis et al, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm"

packet loss rate $p = 1$ per (number of packets sent in W/2 round trips)

$3/4 \times W \times W/2$ packets sent in W/2 round trips

$$p = \frac{3}{8}W^2$$

solving for $W = \sqrt{\frac{8}{3p}}$

average transfer rate $= \frac{3}{4}W = C \cdot \sqrt{1/p}$ (for some $C$)

# emulation

P4 assignment virtual machine

based on *mininet*

creates "virtual" network devices on Linux system

- no actual hardware for network device
- connected to program (implementing P4 switch) and/or other virtual network devices

emulation runs 'as fast as it can':

- performance based on speed of programs talking to simulated devices

could add artificial delays/bandwidht constraints to emulate more realistic networks

- make things more realistic by making things slower?

# emulation constraints

does not reflect real network deployment in important ways

- especially for evaluating congestion control...

performance of one simulated hosts affects others because shared machine

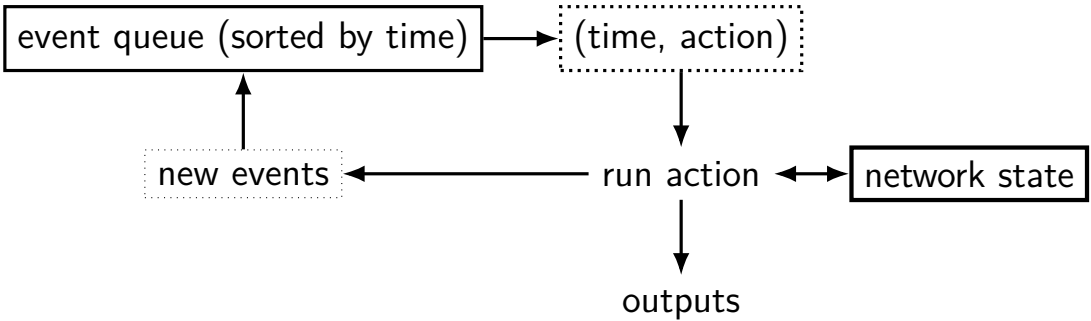
performance will be unrealistically low for large networks

round-trip timing affected a lot by packet processing speeds

speed of packets being sent limited by how fast CPU is

- never going to get to ‘real’ speed of hardware-accelerated switches

# discrete-event simulation



# action example 1

take next packet from send queue for link X

compute whether packet is lost due to error

compute when packet is done transmitting

- schedule new event to handle next packet in queue at that time

compute reception time of packet on other end of link

- schedule new event to handle packet being received at that time

# action example 2

take next packet on link 0 of switch

compute next link for packet

add packet to queue for next link

schedule new events:

- to dequeue from next link (if not scheduled already)

# NS-3

discrete event simulator planned for AIMD assignment

written in C++

(yes, I know it's not the most familiar language)
(obvious alternative simulators aren't in better languages...)

create simulations by writing C++ programs

# an NS-3 event handler

```
bool PointToPointChannel::TransmitStart(
    Ptr<const Packet> p,
    Ptr<PointToPointNetDevice> src,
    Time txTime
) {
  // ...
  uint32_t wire = src == m_link[0].m_src ? 0 : 1;

  Simulator::ScheduleWithContext(
    m_link[wire].m_dst->GetNode()->GetId(),
    txTime + m_delay,
    &PointToPointNetDevice::Receive, m_link[wire].m_dst, p->Copy());

  // Call the tx anim callback on the net device
  m_txrxPointToPoint(p, src, ...)
  return true;
}
```

# an NS-3 event handler

X::Y = Y method/variable of X class

```
bool PointToPointChannel::TransmitStart(
    Ptr<const Packet> p,
    Ptr<PointToPointNetDevice> src,
    Time txTime
) {
  // ...
  uint32_t wire = src == m_link[0].m_src ? 0 : 1;

  Simulator::ScheduleWithContext(
    m_link[wire].m_dst->GetNode()->GetId(),
    txTime + m_delay,
    &PointToPointNetDevice::Receive, m_link[wire].m_dst, p->Copy());

  // Call the tx anim callback on the net device
  m_txrxPointToPoint(p, src, ...)
  return true;
}
```

# an NS-3 event handler

convention: member variables with m_
(C++ member variable ∼ Java instance variable)

```
bool PointToPointChannel::Tra
    Ptr<const Packet> p,
    Ptr<PointToPointNetDevice> src,
    Time txTime
) {
  // ...
  uint32_t wire = src == m_link[0].m_src ? 0 : 1;

  Simulator::ScheduleWithContext(
    m_link[wire].m_dst->GetNode()->GetId(),
    txTime + m_delay,
    &PointToPointNetDevice::Receive, m_link[wire].m_dst, p->Copy());

  // Call the tx anim callback on the net device
  m_txrxPointToPoint(p, src, ...)
  return true;
}
```

# an NS-3 event handler

```
bool PointToPointChannel::TransmitStart(
    Ptr<const Packet> p,
    Ptr<PointToPointNetDevice> src,
    Time txTime
) {
  // ...
  uint32_t wire = src == m_link[0].m_src ? 0 : 1;

  Simulator::ScheduleWithContext(
    m_link[wire].m_dst->GetNode()->GetId(),
    txTime + m_delay,
    &PointToPointNetDevice::Receive, m_link[wire].m_dst, p->Copy());

  // Call the tx anim callback on the net device
  m_txrxPointToPoint(p, src, ...)
  return true;
}
```

setup future event; args:
context (for logging mostly)
time event will trigger
method to run + arguments to pass

# sample NS-3 simulation — setup (1)

ns3/examples/tcp/tcp-bulk-send.cc

```
// nodes = routers or endpoints
NodeContainer nodes;
nodes.Create(2);

// create simulated point-to-point link
    // also supported: multi-access links
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("500Kbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("5ms"));

// setup emulated NICs (which have queues, etc.)
NetDeviceContainer devices;
devices = pointToPoint.Install(nodes);

// setup emulated TCP/IP implementation
InternetStackHelper internet;
internet.Install(nodes);

...
```

# sample NS-3 simulation — setup (2)

ns3/examples/tcp/tcp-bulk-send.cc

```
// Simulated "applications" that send/receive data
BulkSendHelper source("ns3::TcpSocketFactory", InetSocketAddress(i.GetAddress(1),
source.SetAttribute("MaxBytes", UintegerValue(10000000));
ApplicationContainer sourceApps = source.Install(nodes.Get(0));
sourceApps.Start(Seconds(0));
sourceApps.Stop(Seconds(10));
PacketSinkHelper sink("ns3::TcpSocketFactory", InetSocketAddress(Ipv4Address::GetA
ApplicationContainer sinkApps = sink.Install(nodes.Get(1));
sinkApps.Start(Seconds(0));
sinkApps.Stop(Seconds(10));
```

# sample NS-3 simulation — setup (3)

ns3/examples/tcp/tcp-bulk-send.cc

```
AsciiTraceHelper ascii;

// produces text trace file of simulator events
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("tcp-bulk-send.tr"));

// produces PCAP files you can open in Wireshark
pointToPoint.EnablePcapAll("tcp-bulk-send", false);
```

# running example: –help

```
$ ./ns3 run examples/tcp/tcp-bulk-send -- --help
[0/2] Re-checking globbed directories...
ninja: no work to do.
tcp-bulk-send [Program Options] [General Arguments]

Program Options:
    --tracing:   Flag to enable/disable tracing [false]
    --maxBytes:  Total number of bytes for application to send [0]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintVersion:              Print the ns-3 version.
    --PrintHelp:                 Print this help message.
```

# running example

```
$ ./ns3 run examples/tcp/tcp-bulk-send -- --maxBytes=1000000 --tracing=true
[0/2] Re-checking globbed directories...
ninja: no work to do.
Total Bytes Received: 566016

$ wireshark tcp-bulk-send-0-0.pcap
$ wireshark tcp-bulk-send-1-0.pcap
$ wireshark tcp-bulk-send-2-0.pcap
$ wireshark tcp-bulk-send-2-1.pcap
```

---

packet traces for each simulated interface: (host)-(interface).pcap

also .tr file with all events

only 566016 bytes received because 10 second simulation only

500Kbit/s = 62KByte/s @ 10 s = about 620KByte

# TCPSocketState

class to track socket state

- regarldess of congestion control algorithm

includes (notably for us)

- ‘congesiton state’
- congestion window (cwnd)
- slow start threshold (ssthresh)

# TCP congestion states

OPEN — normal

DISORDER — dupacks/SACKs below threshold for recovery

RECOVERY — retransmitting due to dup-ACK/simple SACK

temporarily inflated window to account for dropped packets

LOSS — retransmitting due to timeout/etc.

# configurable TCP behavior

TCPRecoveryOps: handle recovery

- EnterRecovery (called on start of retransmitting)
  - default: cwnd $\rightarrow$ ssthresh
- DoRecovery (called for ACK when retransmitting)
  - default: cwnd $\rightarrow$ cwnd $+$ 1 packet
- ExitRecovery (called when 'back to normal')
  - default: cwnd $\rightarrow$ ssthresh

*TCPCongestionOps*:

- IncreaseWindow (called when new segments ACKed)
- GetSsThresh (called after loss)

# assingment TCP changes

will customize a `MyTcpCongestionOps`

...which inherits from TCPNewReno

you can customize increase (IncreaseWindow) and decrease (GetSsThresh)

# IncreaseWindow

```
void
TcpNewReno::IncreaseWindow(Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
    NS_LOG_FUNCTION(this << tcb << segmentsAcked);

    if (tcb->m_cWnd < tcb->m_ssThresh)
    {
        segmentsAcked = SlowStart(tcb, segmentsAcked);
    }

    if (tcb->m_cWnd >= tcb->m_ssThresh)
    {
        CongestionAvoidance(tcb, segmentsAcked);
    }
}
```

# CongestionAvoidance

```
void
TcpNewReno::CongestionAvoidance(Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
    NS_LOG_FUNCTION(this << tcb << segmentsAcked);

    if (segmentsAcked > 0)
    {
        double adder =
            static_cast<double>(tcb->m_segmentSize * tcb->m_segmentSize) / tcb->m_
        adder = std::max(1.0, adder);
        tcb->m_cWnd += static_cast<uint32_t>(adder);
        ...
    }
}
```

additive increase (plus 1 segment per cwnd)

# SlowStart

```
uint32_t
TcpNewReno::SlowStart(Ptr<TcpSocketState> tcb, uint32_t segmentsAcked)
{
    NS_LOG_FUNCTION(this << tcb << segmentsAcked);

    if (segmentsAcked >= 1)
    {
        tcb->m_cWnd += tcb->m_segmentSize;
        ...
        return segmentsAcked - 1;
    }

    return 0;
}
```

multiplicative increase — doubling

# GetSsThresh

```
uint32_t
TcpNewReno::GetSsThresh(Ptr<const TcpSocketState> state, uint32_t bytesInFlight)
{
    NS_LOG_FUNCTION(this << state << bytesInFlight);

    return std::max(2 * state->m_segmentSize, bytesInFlight / 2);
}
```

multiplicative decrease (bytesInFlight / 2)

# aside: more advanced congestion hooks

TcpCongestionOps also has access to

- ECN (explicit congestion notification) info
- timestamps
- exactly when duplicate ACKs below threshold/ recovery/etc. occurs

used by more advanced TCP implementatoins

# assignment preview

start with `MyTcpCongestionOps`

- basically copy of NewRenoTcp

setup to take 'mode' parameter

- mode 0: default behavior
- mode 1 (added by you): take more bandwidth
- mode 2 (added by you): take less bandwidth

# NS-3 logging

NS3 logging tools

`https://www.nsnam.org/docs/manual/html/logging-asserts.html`

doing logging:

```
// at top of file
NS_LOG_COMPONENT_DEFINE("SomeLabel");
...
// when you want to log
NS_LOG_INFO("value of x=" << x);
    // INFO could be DEBUG, ERROR, ... too
```

enabling logging:

in C++: `LogComponentEnable("SomeLabel", LOG_LEVEL_DEBUG)`

# backup slides