University of Nottingham
UK | CHINA | MALAYSIA

COMP1047: Systems and Architecture

Dr. Fazl Ullah (Khan)

AY2023-24, Spring Semester
Week 9
Computer Networks:
TCP/IP Protocol Suite & Performance Evaluation

Introduction

- Most of the slides are based on the Books

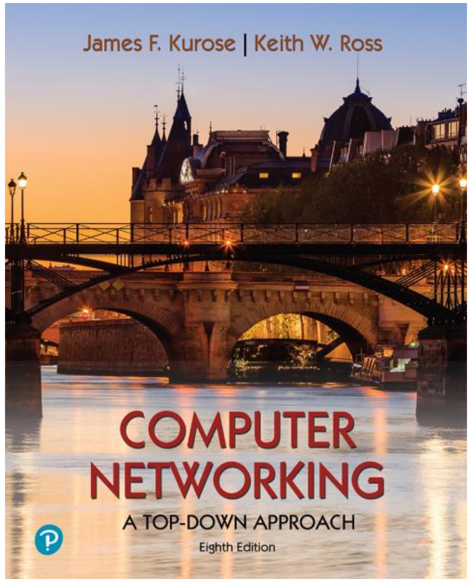
1. ***Computer Networking: A Top-Down Approach***

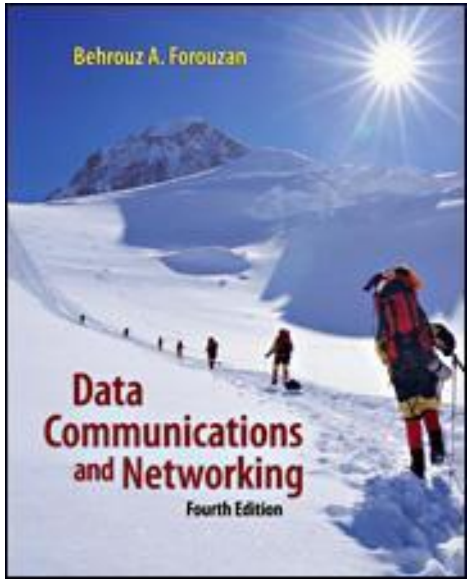
   8th edition by Jim Kurose, Keith Ross

and

2. ***Data Communication and Networking***

   4th edition by Behrouz A. Forouzan





TCP Protocol and Performance Evaluation

Learning Outcomes

- Understand the Services of TCP
  - Connection Management
  - Congestion control
- Analyzing Network Performance
  - Delay
  - Throughput
  - Packets drop/delivery

Roadmap

- TCP: Connection-oriented Protocol
  - Flow Control (Block TCP Problem)
  - TCP Acks. Seq. No. and RTT
  - TCP Connection Management
- Congestion Control
  - Causes
  - Costs
- Performance Analysis
  - Performance Evaluation Approaches
  - NS2
  - Otcl scripts for NS2

Overview

- *Recap from last Week*
  - Block TCP Flow Problem
    - Deadlock like situation
- TCP: Connection-oriented protocol
    - Sequence Number
    - Ack Number
    - RTT and timeout
- Connection Management
    - Handshake
    - Closing TCP Connection
- TCP: Connection-oriented protocol
    - Congestion control
    - Causes of congestion
    - Cost of congestion
- Network Performance Analysis
    - Approaches
    - NS-2
    - Otcl scripts

Recap from the Last Week

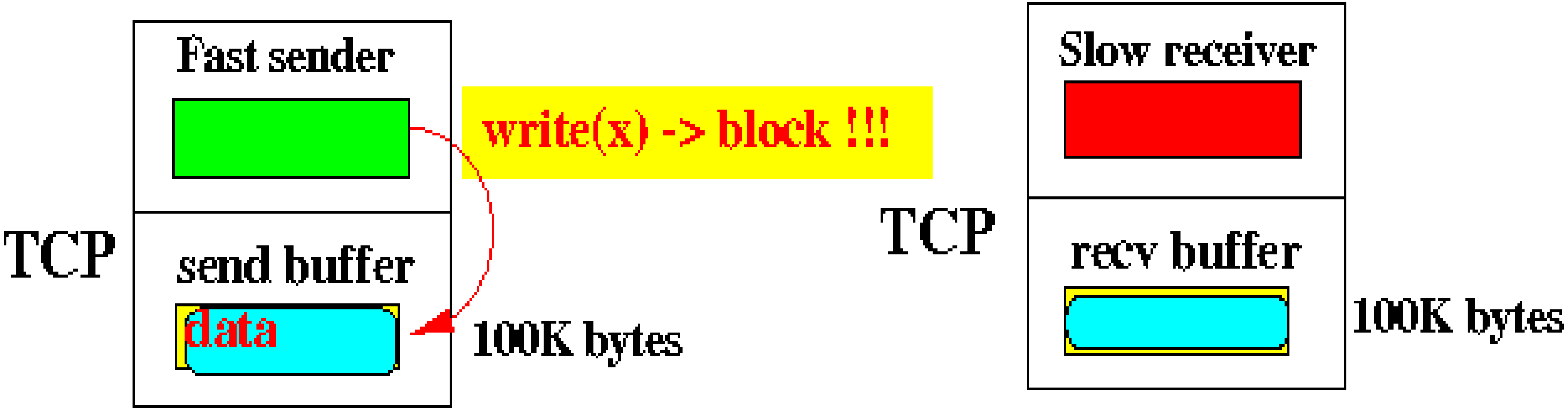
Last Week, we discussed

- Principles behind Transport layer
  - Services
  - Actions
  - Protocols
- UDP: Connectionless protocol
  - Message Format
  - Multiplexing
  - Checksum
- TCP: Connection-oriented protocol
  - Message Format
  - End-to-End Reliable Communication
  - Flow Control

Any question in previous lecture?

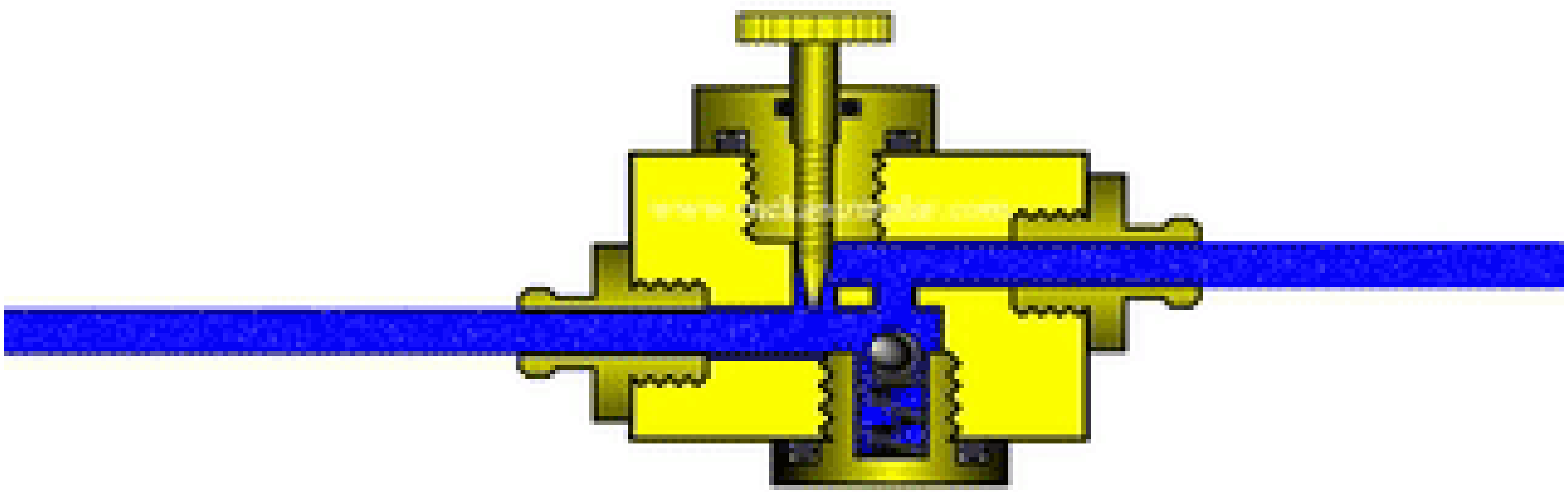
Flow Control- Example From Last Week

- If the **sending application process** continues sending, the **send buffer will fill up**
- A **subsequent `write()` call** will cause the **sending application process** to **block**



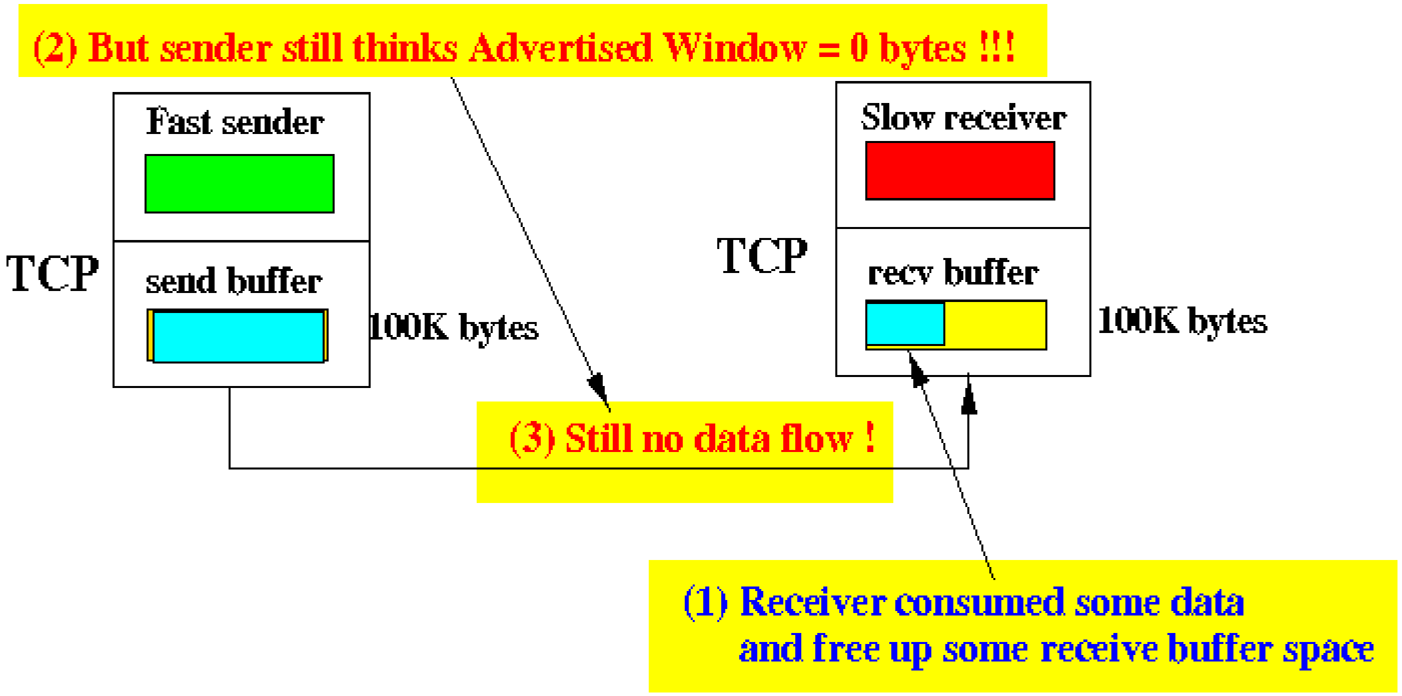
- Now the **faster sending application process** has been **successfully throttle**...

Flow Control



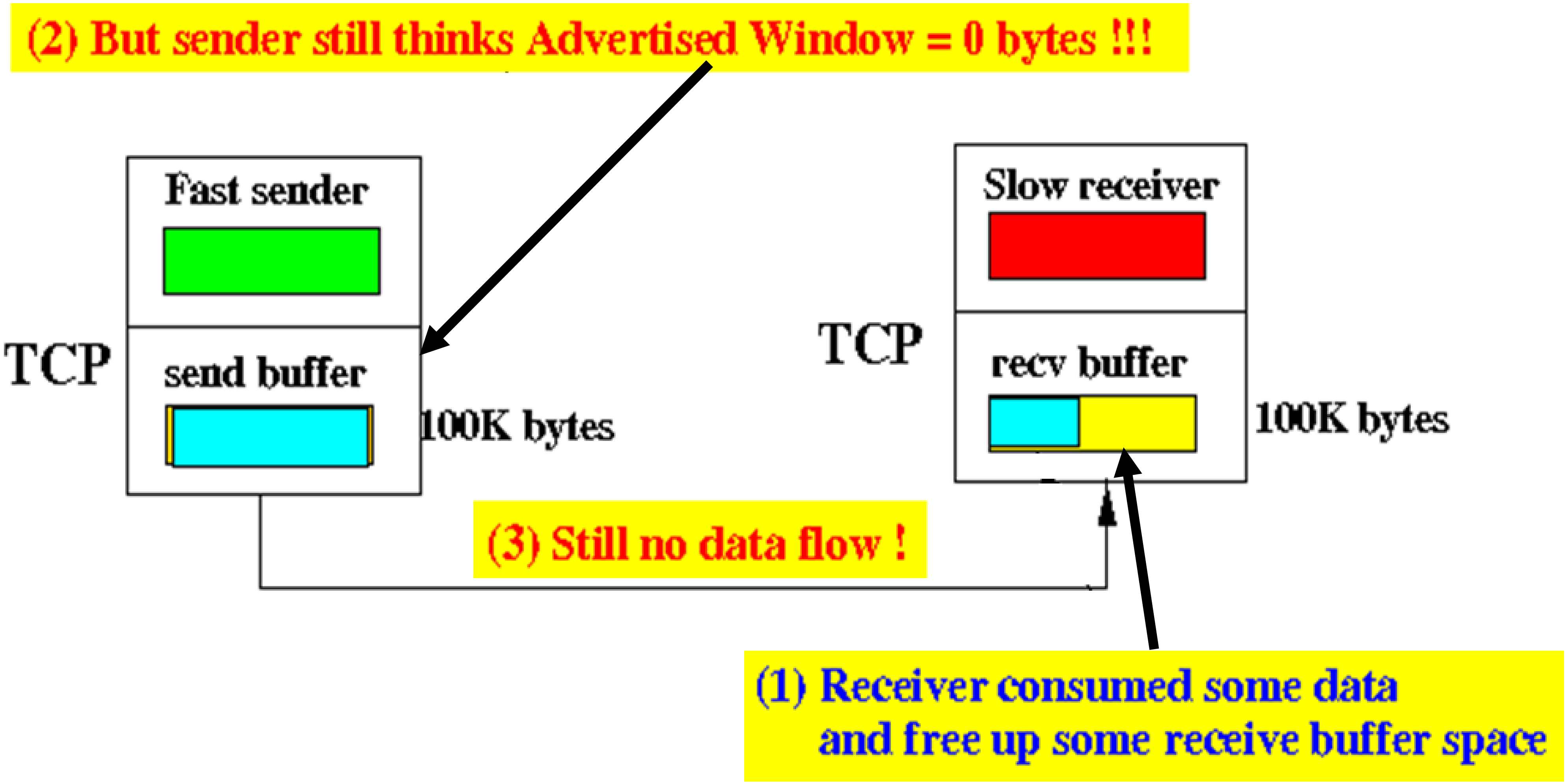
The Block TCP flow problem

- ***Unforeseen consequence* of Flow Control**
- **Advertised window size = 0**, the **sender can *not* send any packet**
- **Consequently**, the **receiver** may ***not* send any packet** to the **sender**
- **As a result**, the **advertised window size** of the **sender** will **remain ZERO**
- We have a "**dead-lock**" **like** situation



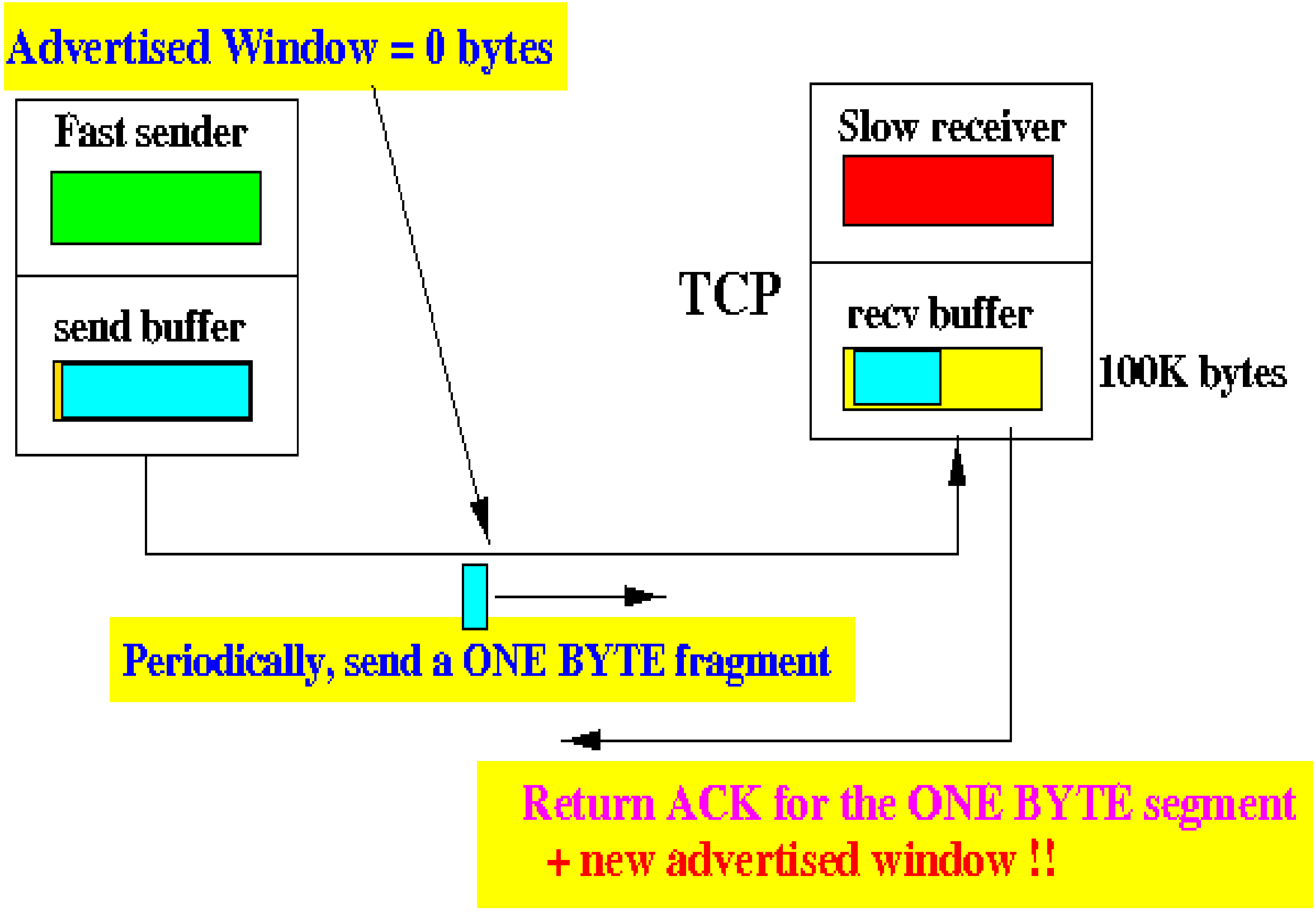
The Block TCP flow problem

- We have a “**dead-lock**” **like** situation called the TCP flow Problem



The Block TCP flow problem

- At sender, **Adv. Win. Size = 0** and it **has some data in the *send buffer***
- It will ***periodically* send a *one byte* TCP segment** to **trigger a response** from the **receiver**



- The **ACK** for the **byte size probe TCP segment** will contain the **new (non-zero) value** of the **Adv. Win. Size**
- the **sender** can use it to **pace its transmissions**

TCP Sequence Numbers & ACKs

Sequence numbers:

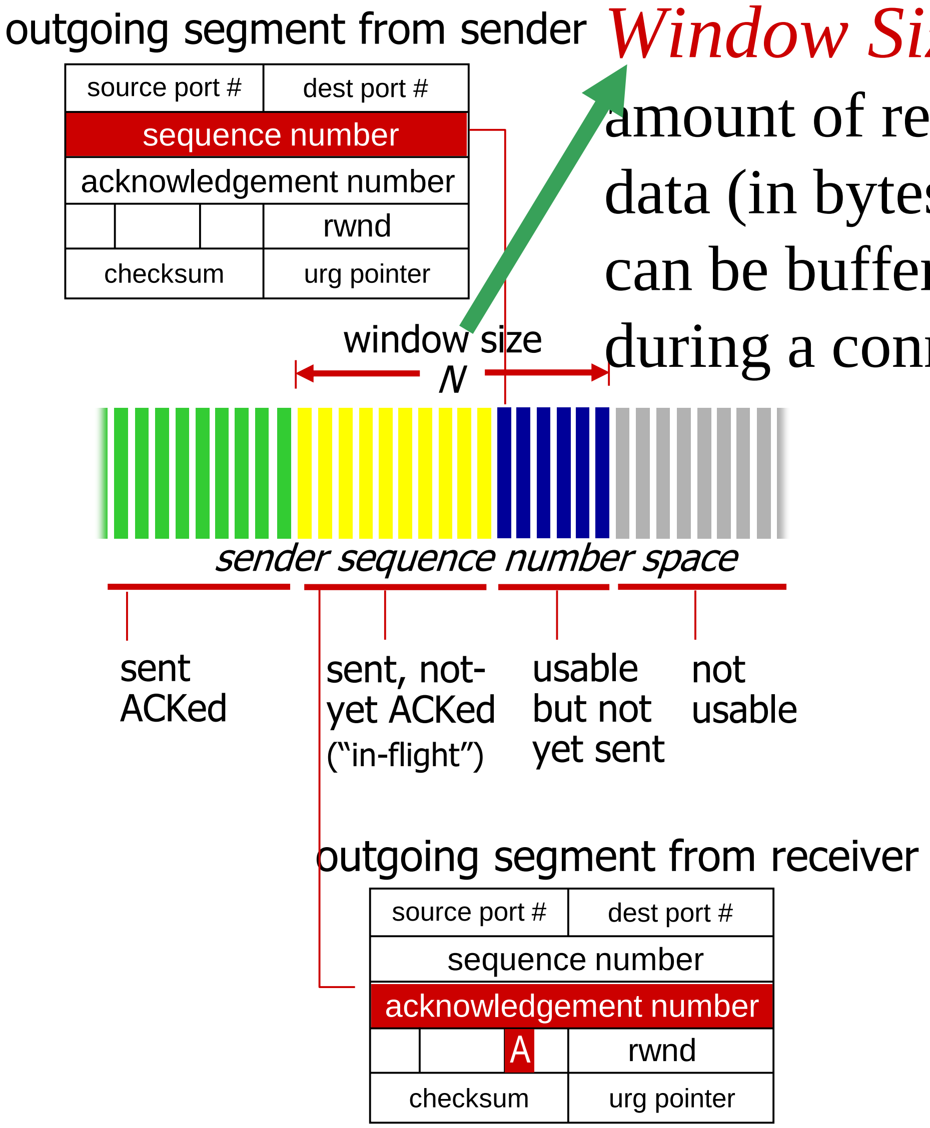
- byte stream "number" of first byte in segment's data

Acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK

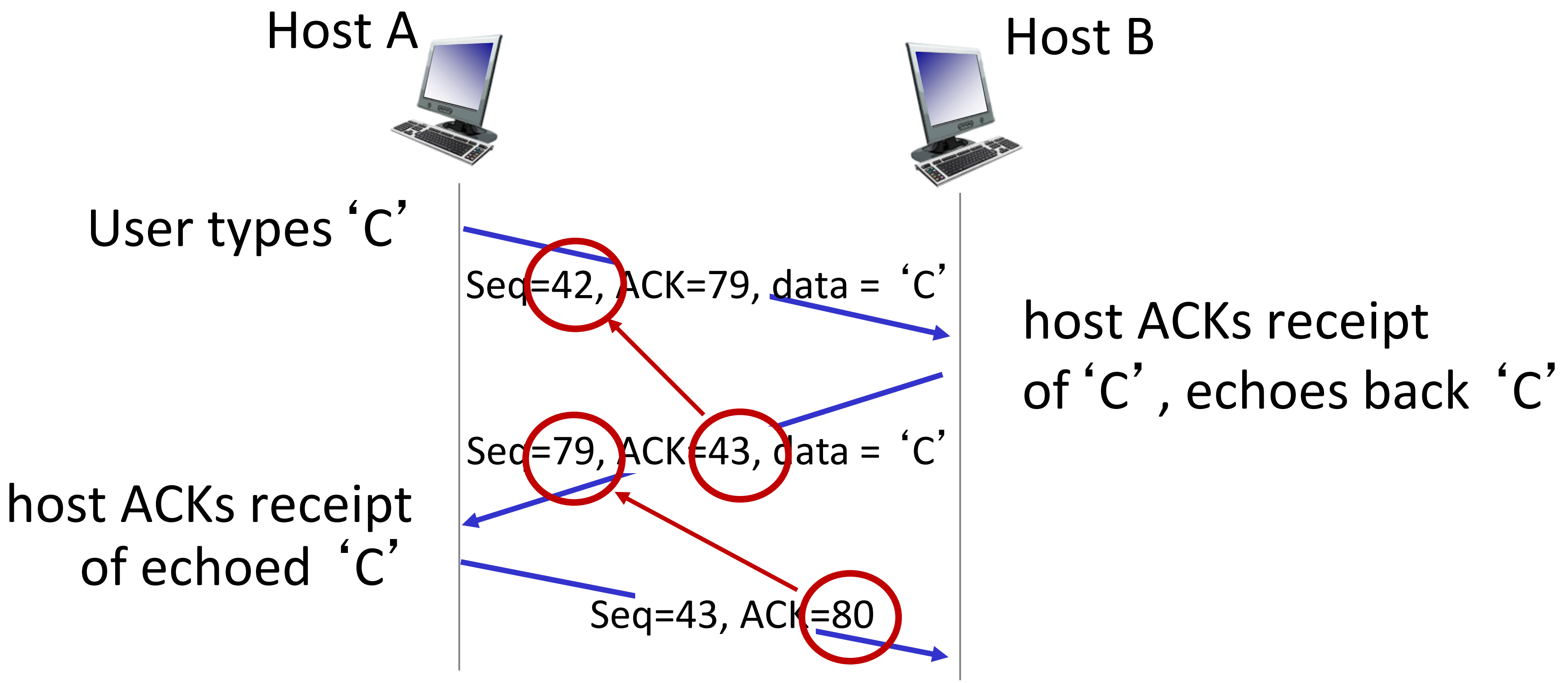
Q: how receiver handles out-of-order segments

- *A:* TCP spec doesn't say, up to implementor



Window Size: the amount of receive data (in bytes) that can be buffered during a connection

TCP Sequence Numbers & ACKs



simple telnet scenario

TCP round trip time, timeout

Q: How to set TCP timeout value?

- longer than RTT, but RTT varies!
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

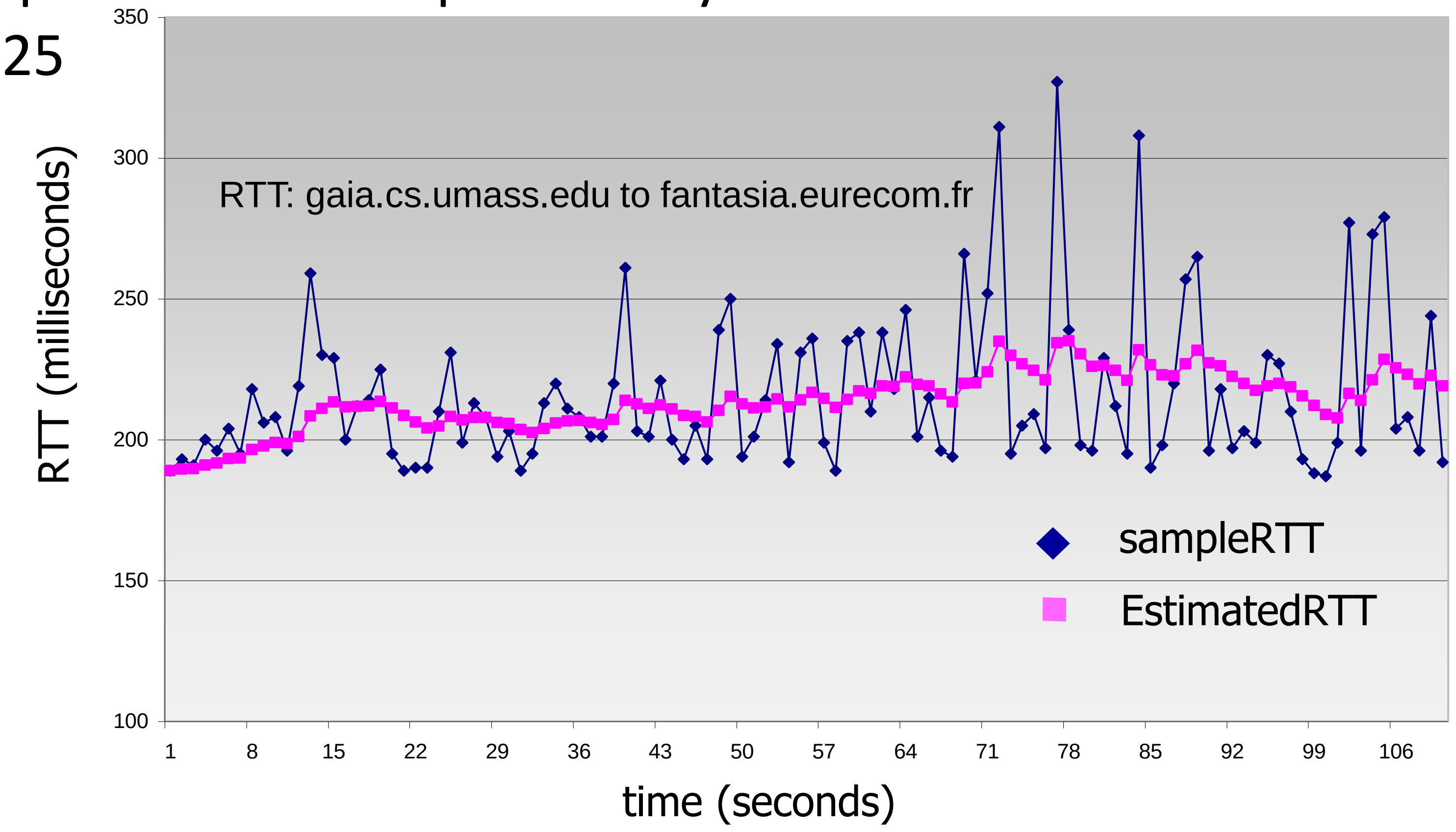
Q: How to estimate RTT?

- SampleRTT: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- SampleRTT will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current SampleRTT

TCP round trip time, timeout

```
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT
```

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



TCP round trip time, timeout

- timeout interval: **EstimatedRTT** plus “safety margin”
  - large variation in **EstimatedRTT**: want a larger safety margin

$$\texttt{TimeoutInterval = EstimatedRTT + 4*DevRTT}$$

estimated RTT

“safety margin”

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

$$\texttt{DevRTT = (1-}\beta\texttt{)*DevRTT + }\beta\texttt{*|SampleRTT-EstimatedRTT|}$$

(typically, β = 0.25)

TCP Sender (simplified)

event: data received from application

- create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: **TimeOutInterval**

event: timeout

- retransmit segment that caused timeout
- restart timer

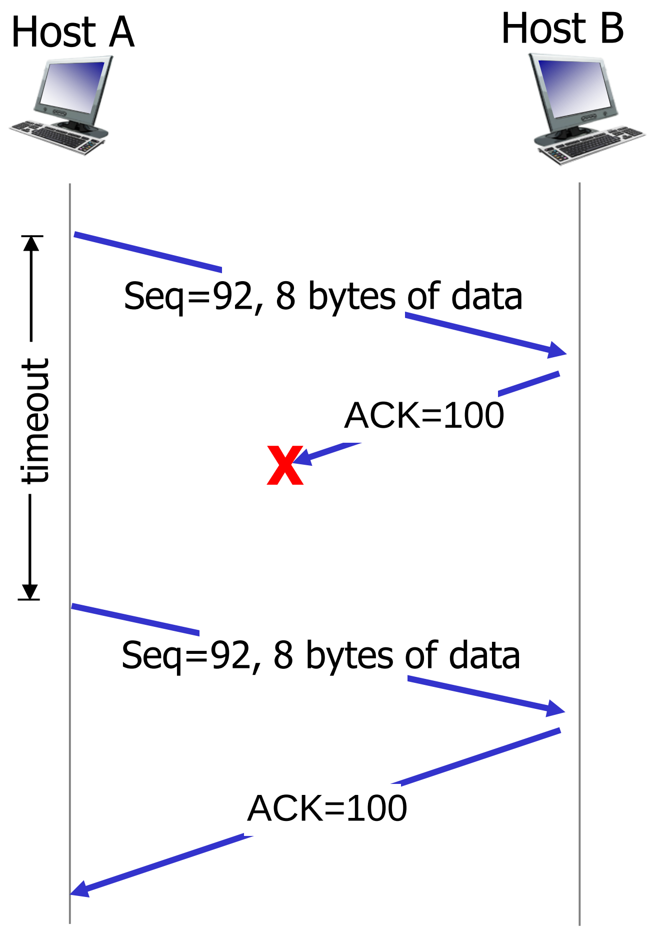
event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

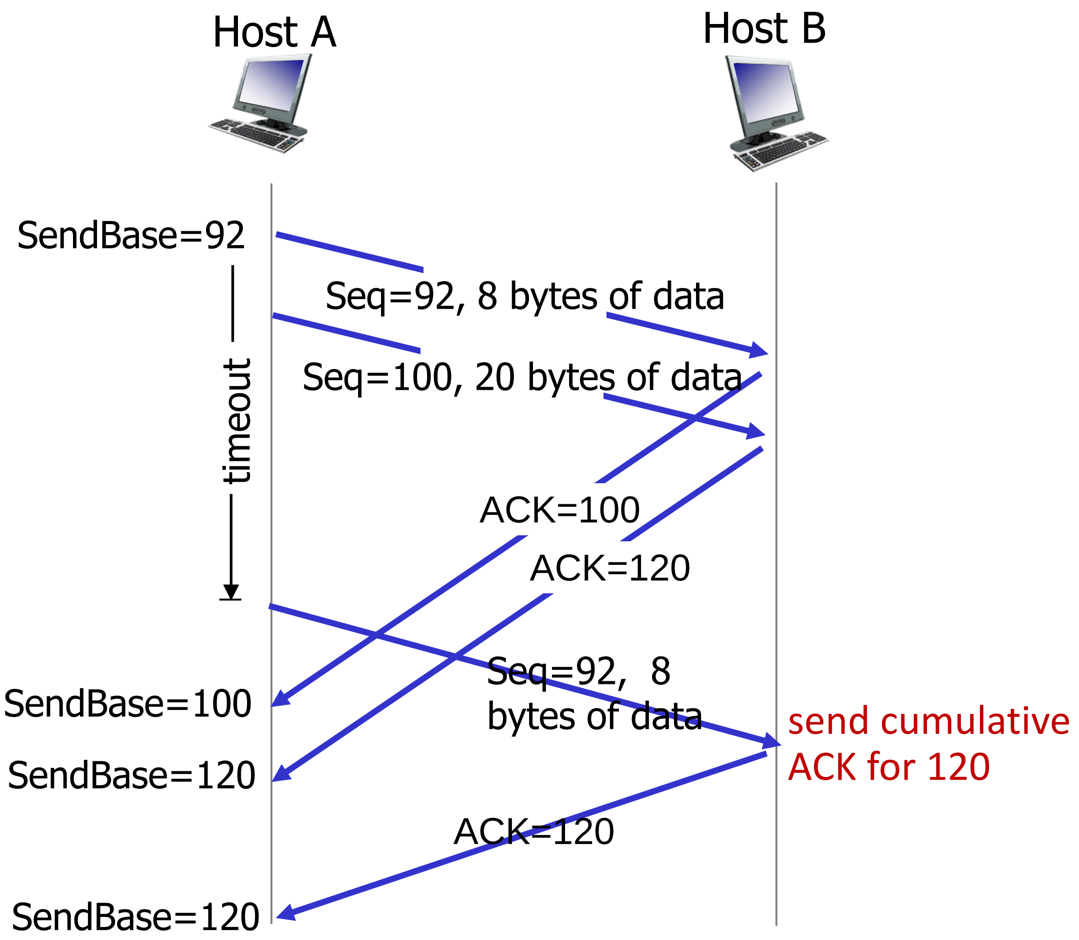
TCP Receiver: ACK generation

Event at receiver	*TCP receiver action*

TCP: retransmission scenarios

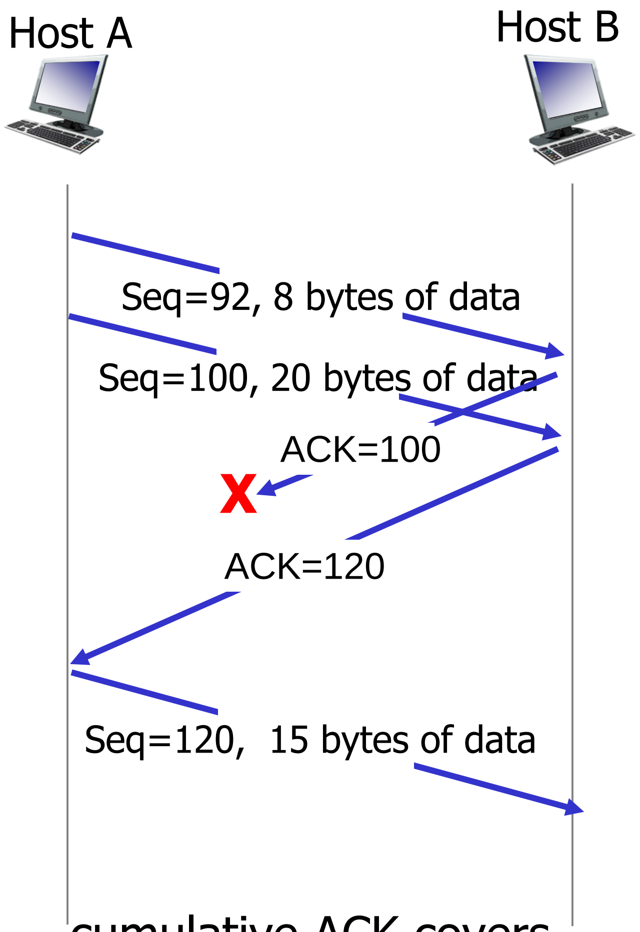


lost ACK scenario



premature timeout

TCP: retransmission scenarios



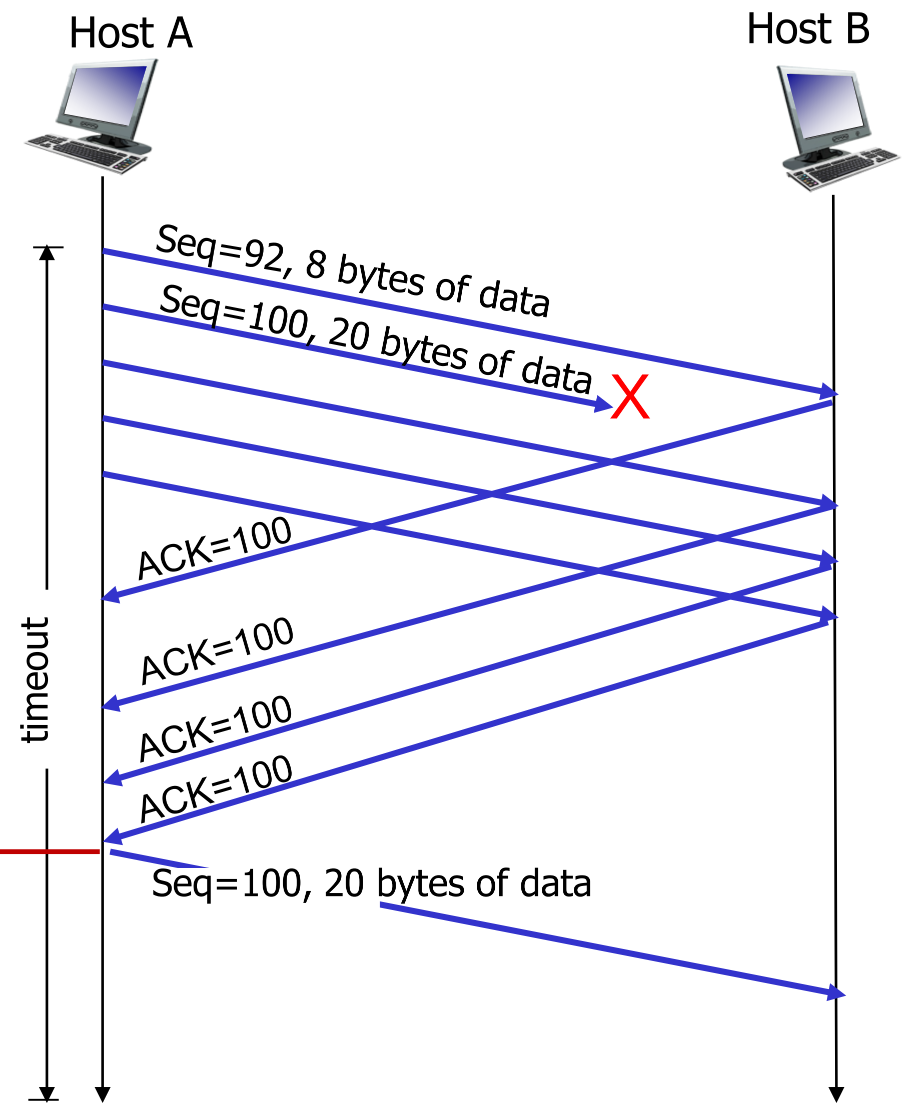
cumulative ACK covers for earlier lost ACK

TCP fast retransmit

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

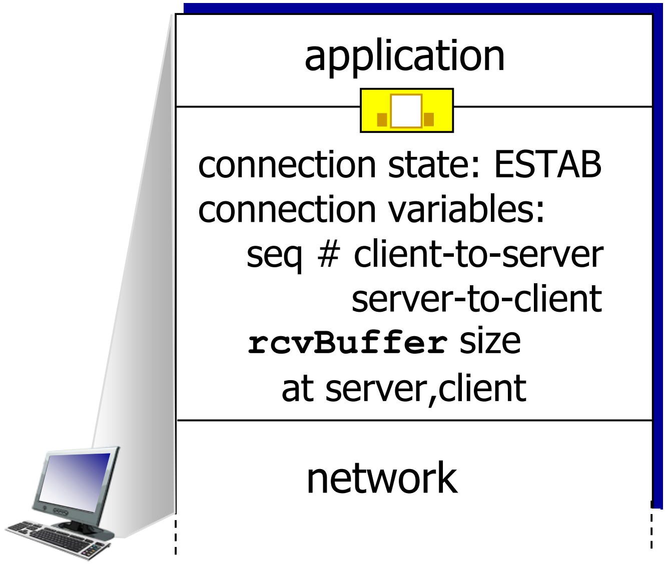


Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

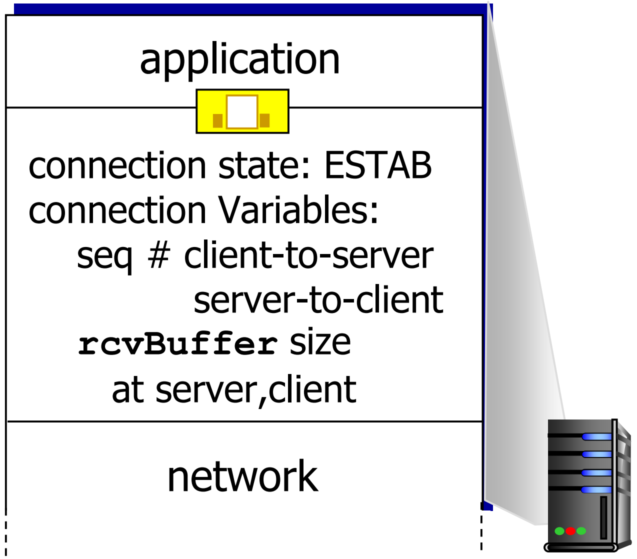
TCP connection management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., starting seq #s)



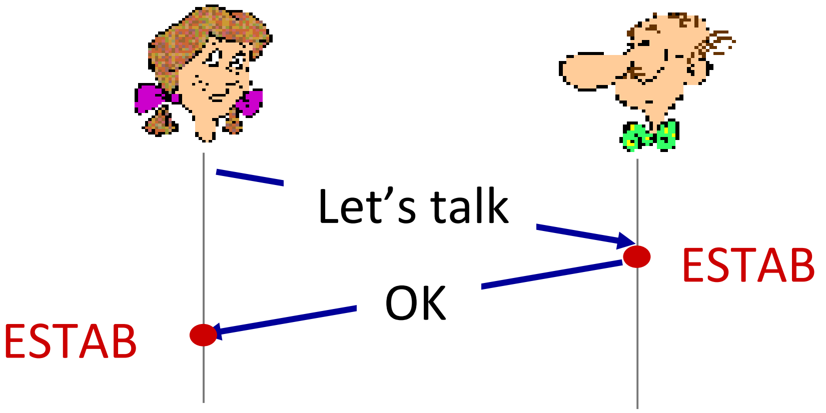

```
Socket clientSocket =
  newSocket("hostname","port number");
```

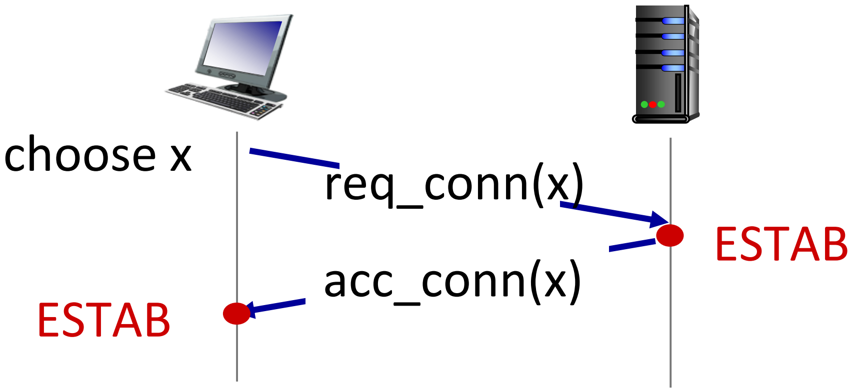



```
Socket connectionSocket =
  welcomeSocket.accept();
```

Agreeing to establish a connection

2-way handshake:

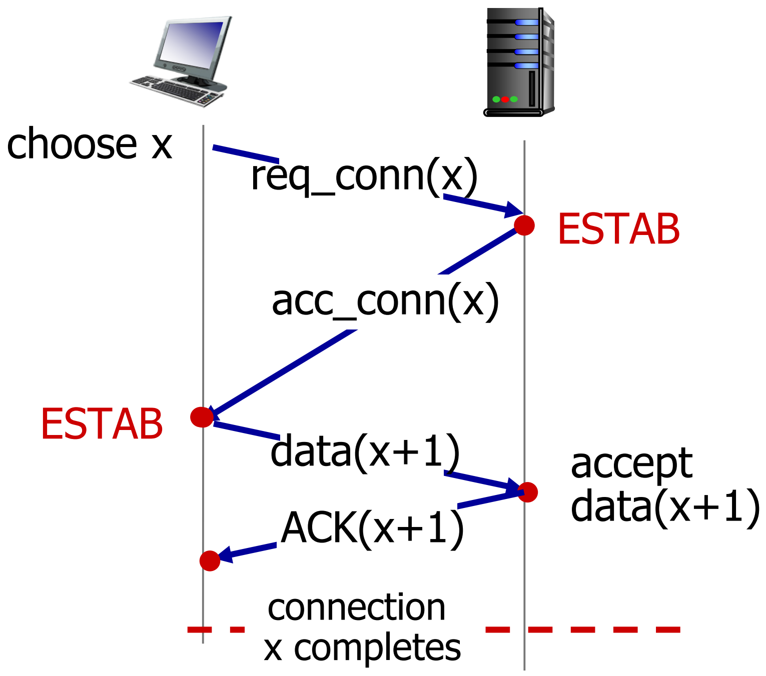




Q: will 2-way handshake always work in network?

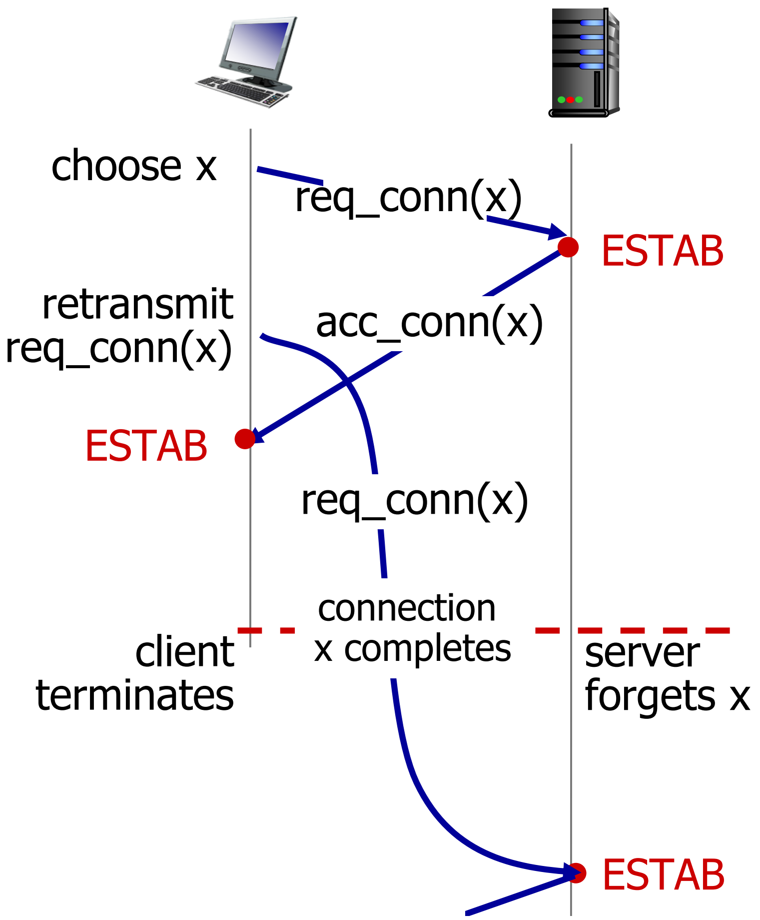
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way handshake scenarios



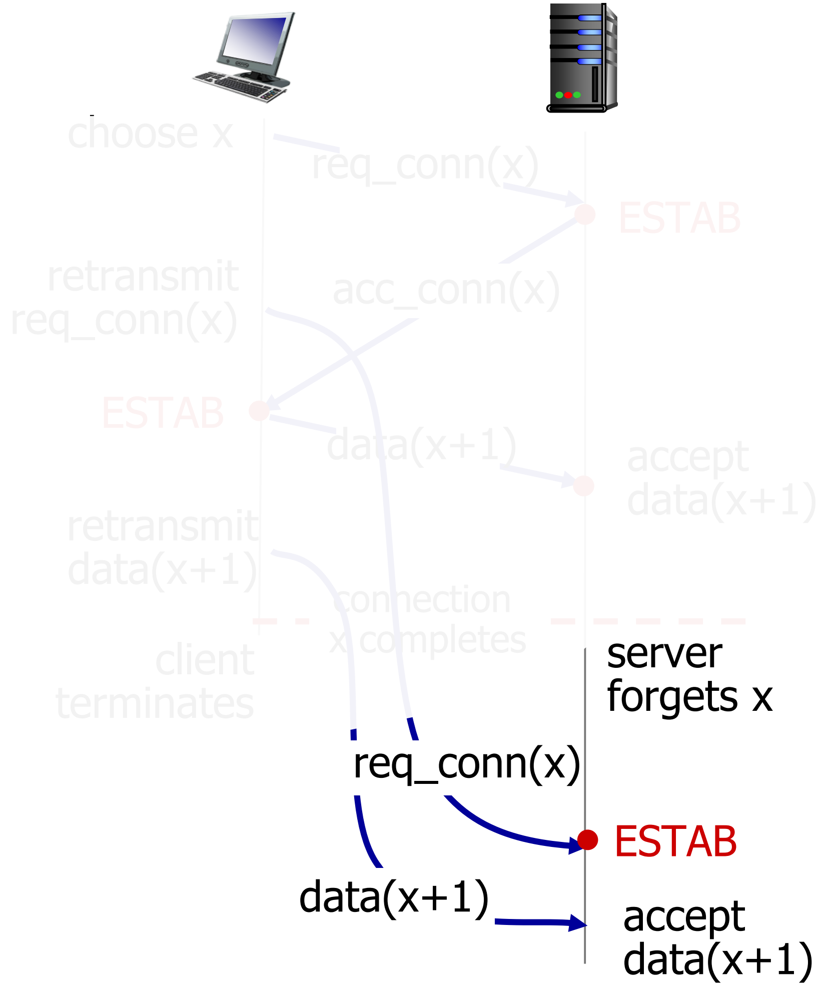
No problem!

2-way handshake scenarios



Problem: half open connection! (no client)

2-way handshake scenarios



Problem: dup data accepted!

TCP 3-way handshake

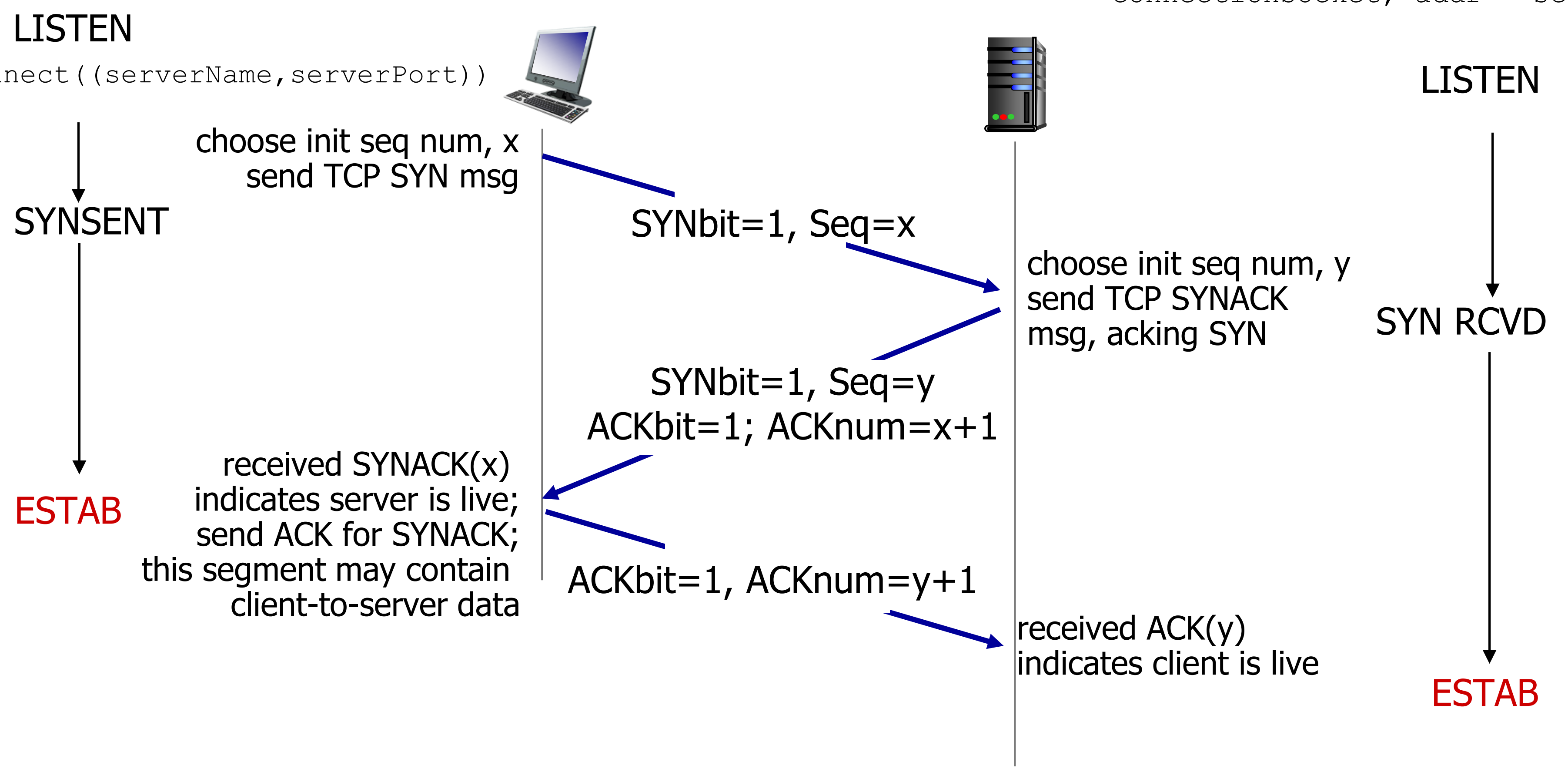
Client state

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

Server state

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
connectionSocket, addr = serverSocket.accept()
```

```
clientSocket.connect((serverName,serverPort))
```

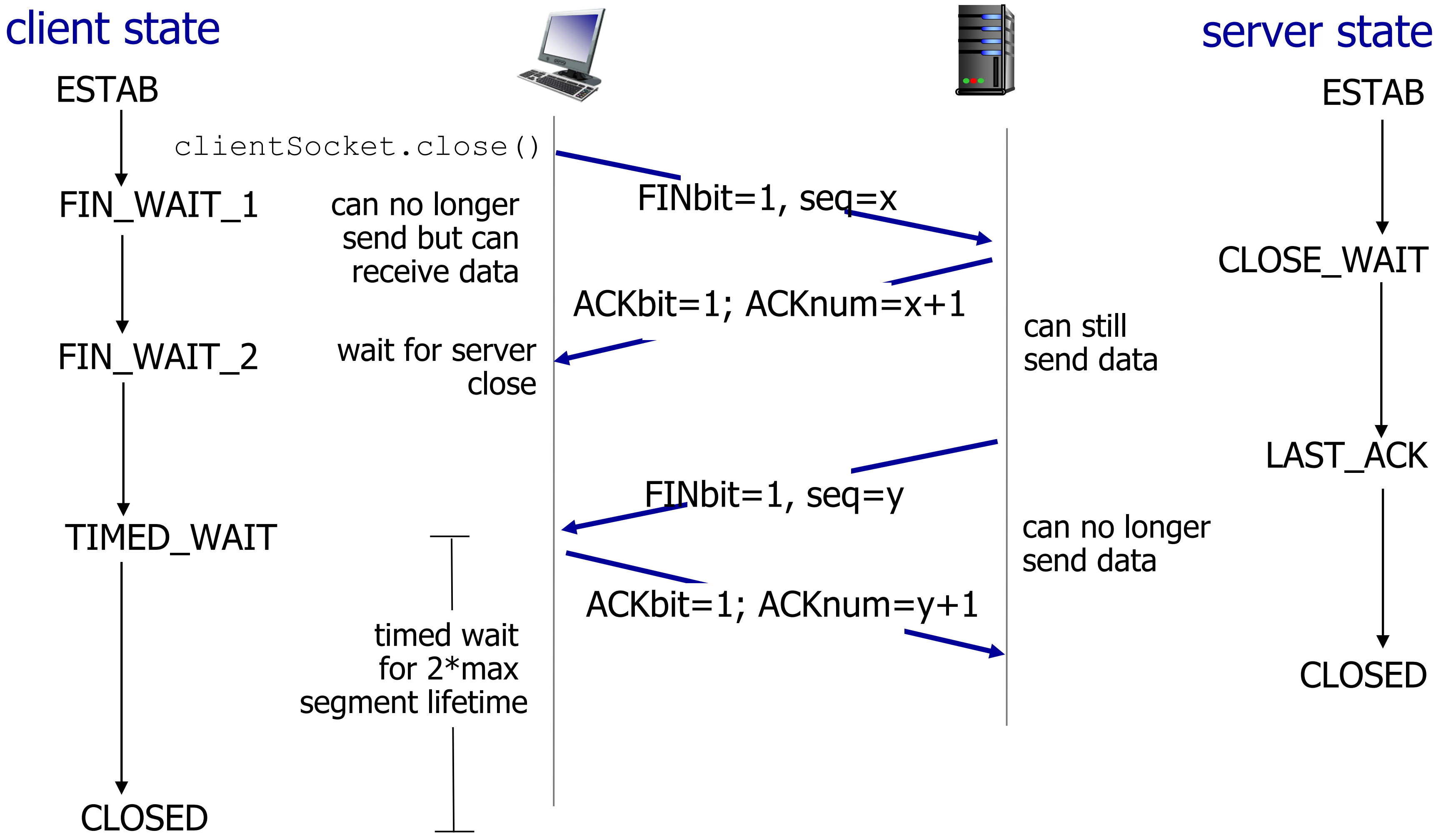


A human 3-way handshake protocol

Closing a TCP connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

Closing a TCP connection



Overview

- Block TCP Flow Problem
  - Deadlock like situation
- TCP: Connection-oriented protocol
  - Sequence Number, Ack Number
  - RTT and timeout
- Connection Management
  - Handshake
  - Closing TCP Connection
- TCP: Connection-oriented protocol
  - Congestion control
  - Causes and Cost of congestion
- Network Performance Analysis
  - Approaches
  - NS-2 and Otcl scripts

Any question?

Principles of Congestion Control

Congestion

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!

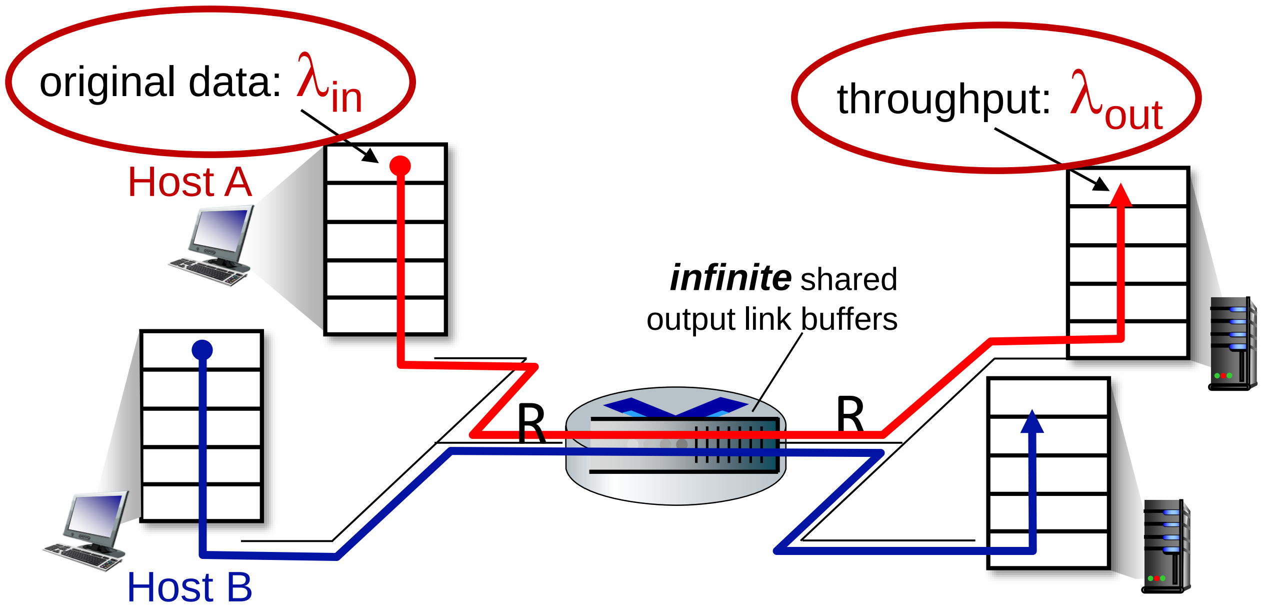
congestion control: too many senders, sending too fast

flow control: one sender too fast for one receiver

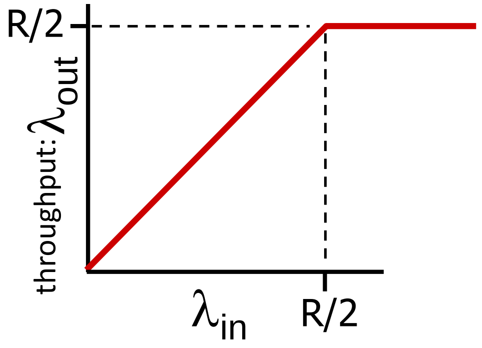
Causes/costs of congestion: scenario 1

Simplest scenario:

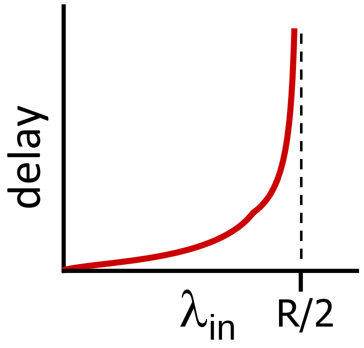
- one router, infinite buffers
- input, output link capacity: R
- two flows
- no retransmissions needed



Q: What happens as arrival rate λ_{in} approaches R/2?



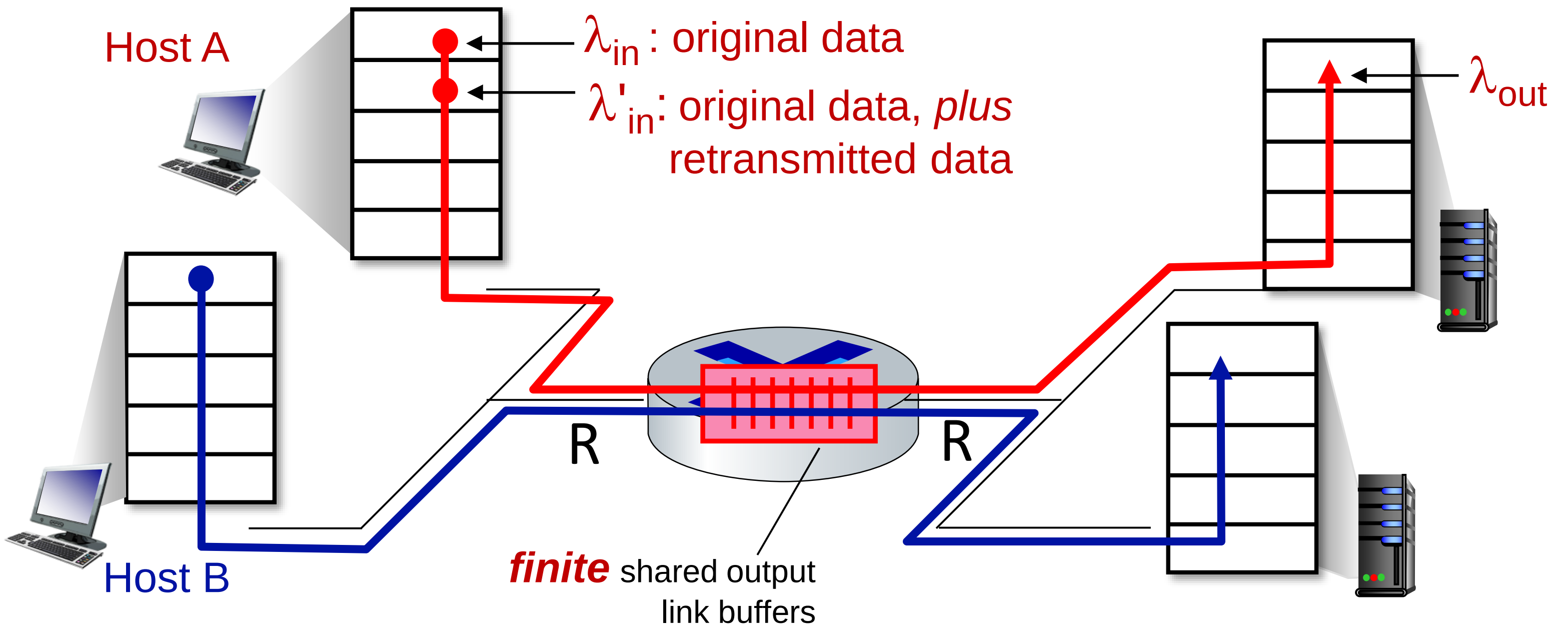
maximum per-connection throughput: R/2



large delays as arrival rate λ_{in} approaches capacity

Causes/costs of congestion: scenario 2

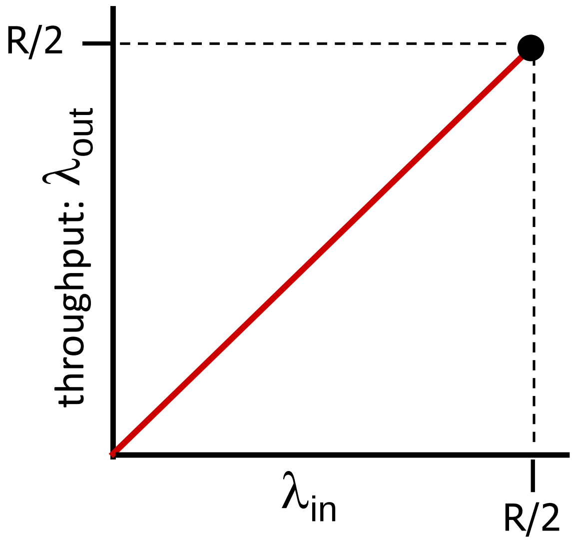
- one router, *finite* buffers
- sender retransmits lost, timed-out packet
  - application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$

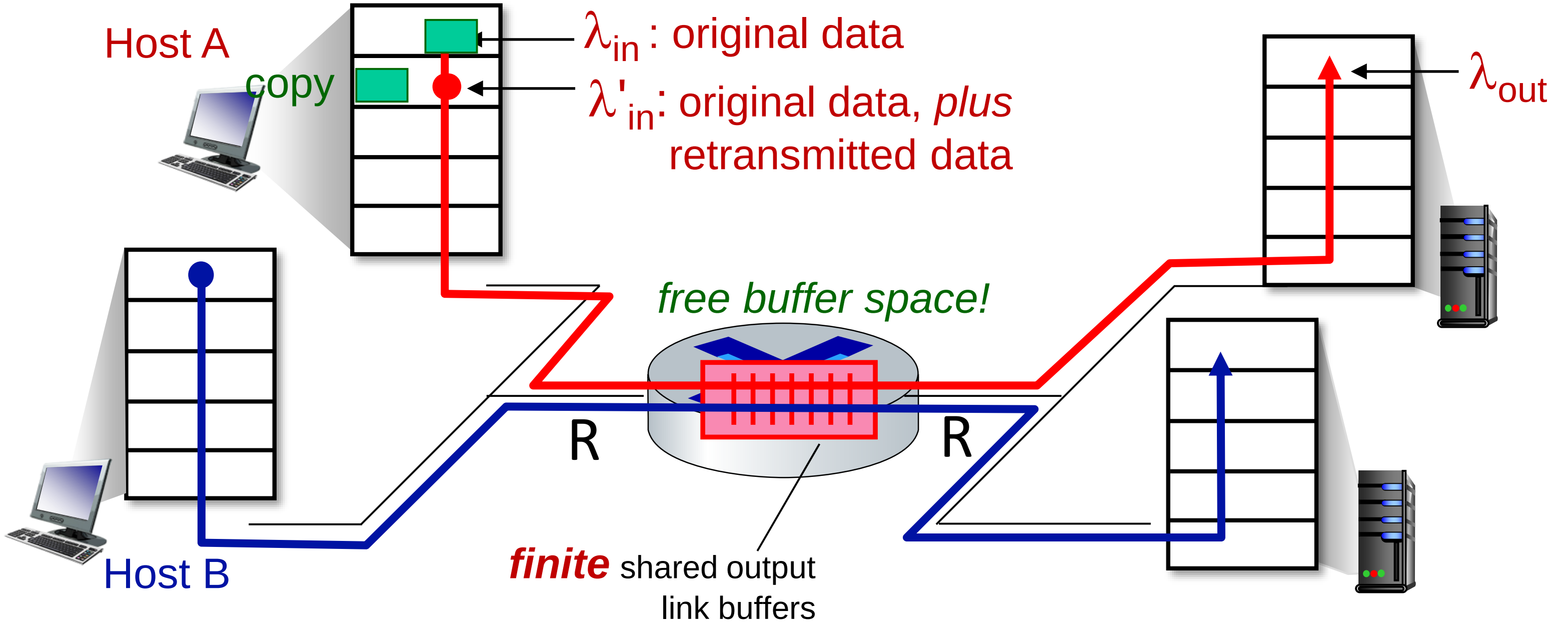


Causes/costs of congestion: scenario 2

Idealization: perfect knowledge

- sender sends only when router buffers available

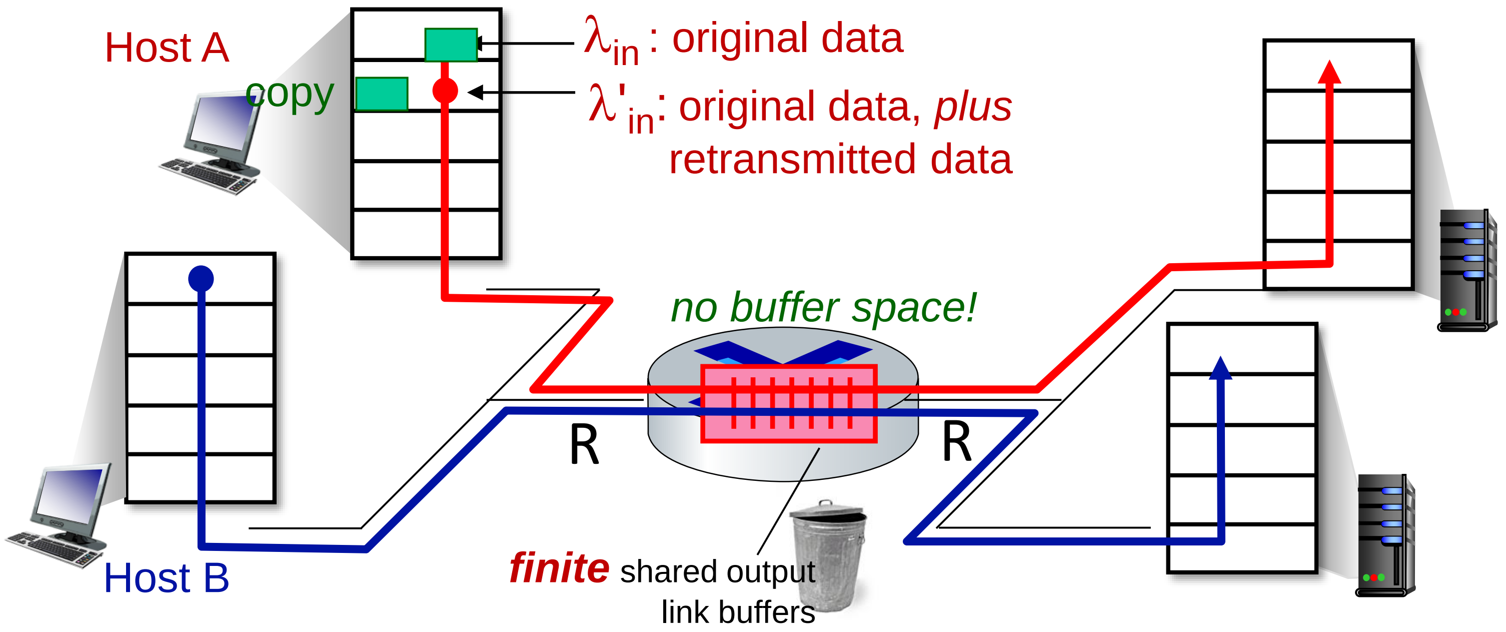




Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

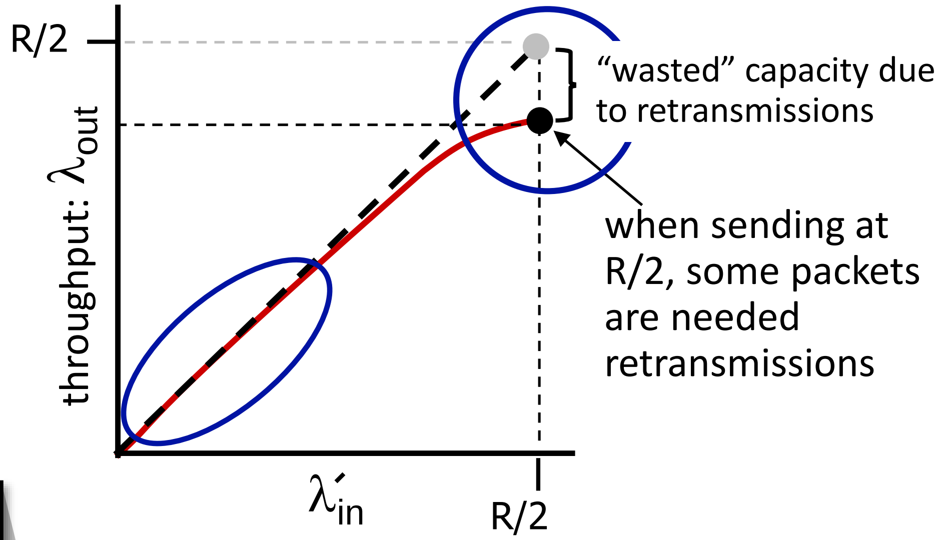
- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

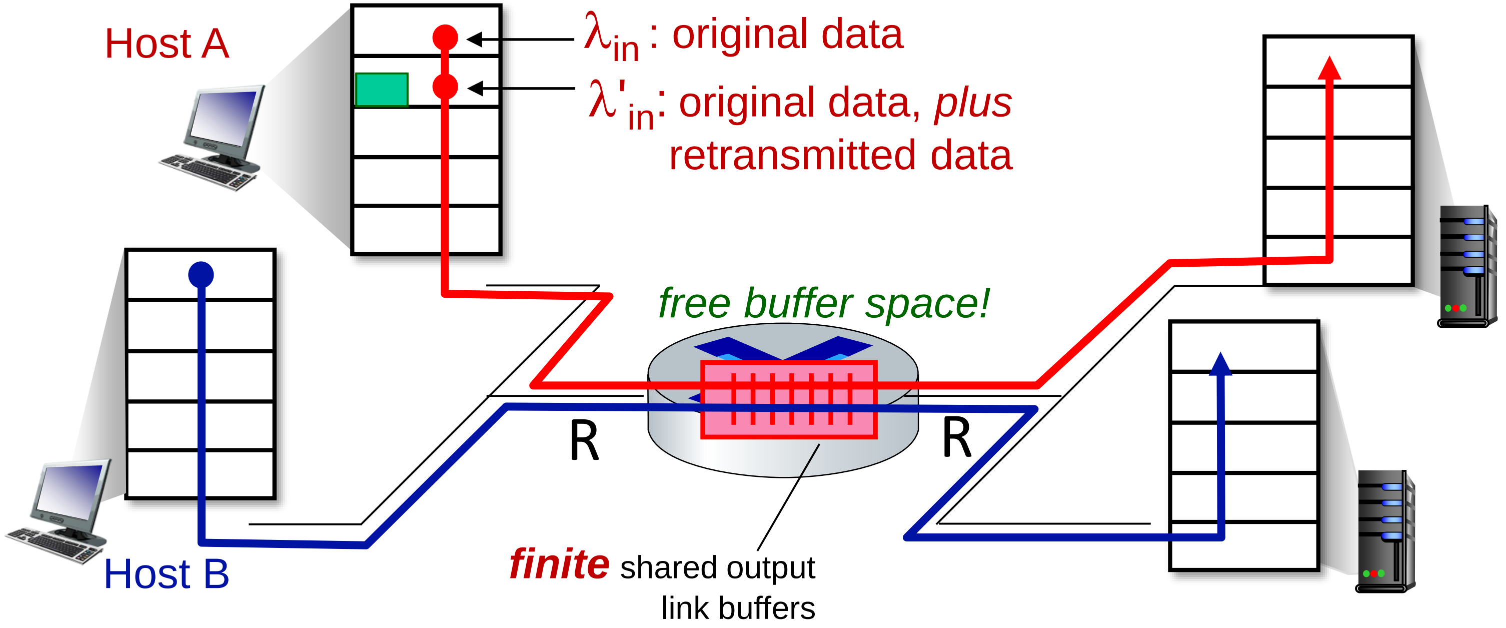


Causes/costs of congestion: scenario 2

Idealization: *some* perfect knowledge

- packets can be lost (dropped at router) due to full buffers
- sender knows when packet has been dropped: only resends if packet *known* to be lost

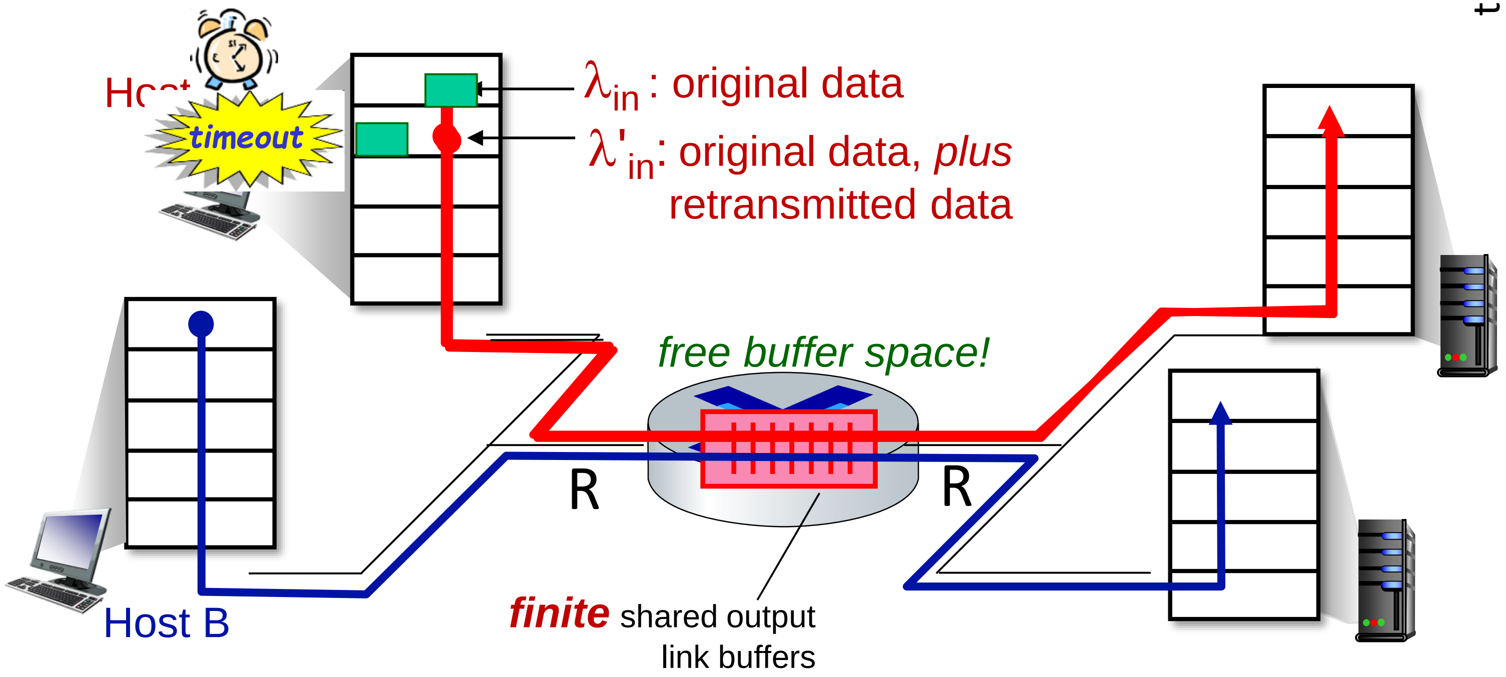


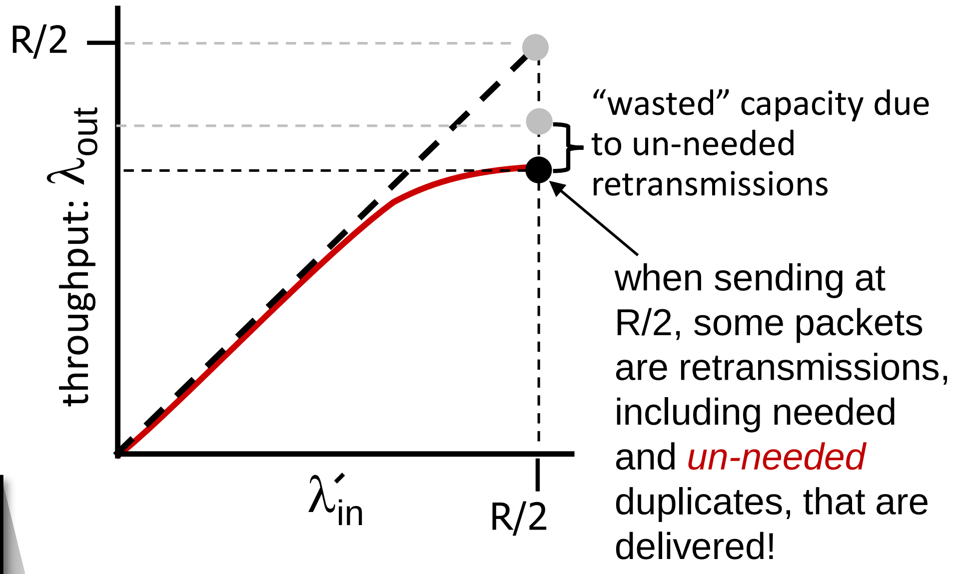


Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

- packets can be lost, dropped at router due to full buffers – requiring retransmissions
- but sender times can time out prematurely, sending *two* copies, *both* of which are delivered

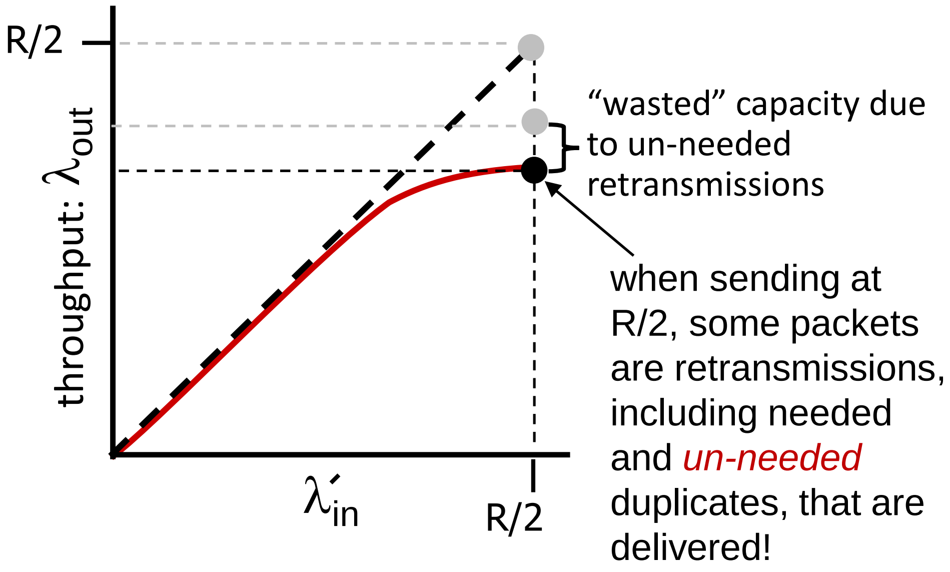




Causes/costs of congestion: scenario 2

Realistic scenario: *un-needed duplicates*

- packets can be lost, dropped at router due to full buffers – requiring retransmissions
- but sender times can time out prematurely, sending *two* copies, *both* of which are delivered



"costs" of congestion:

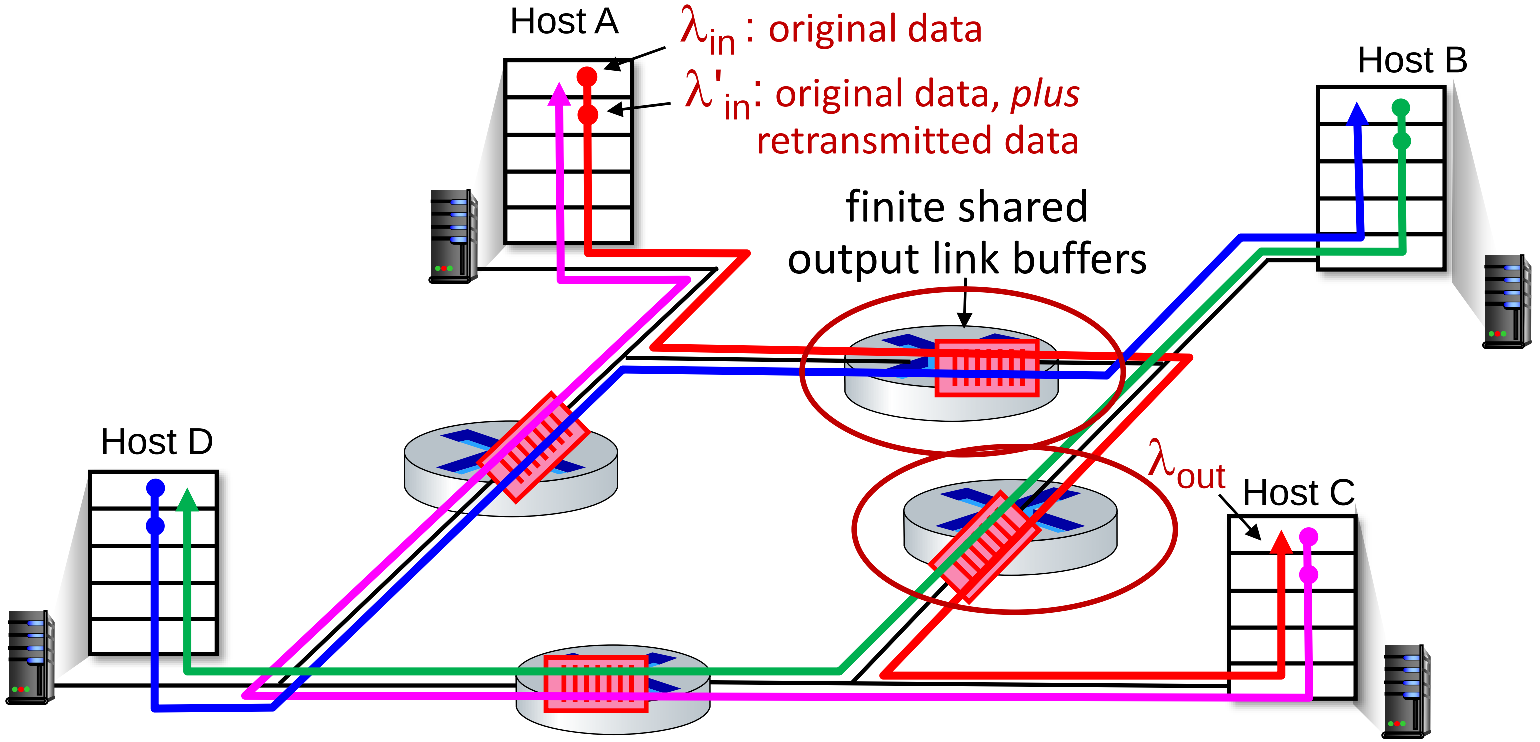
- more work (retransmission) for given receiver throughput
- unneeded retransmissions: link carries multiple copies of a packet
  - decreasing maximum achievable throughput

Causes/costs of congestion: scenario 3

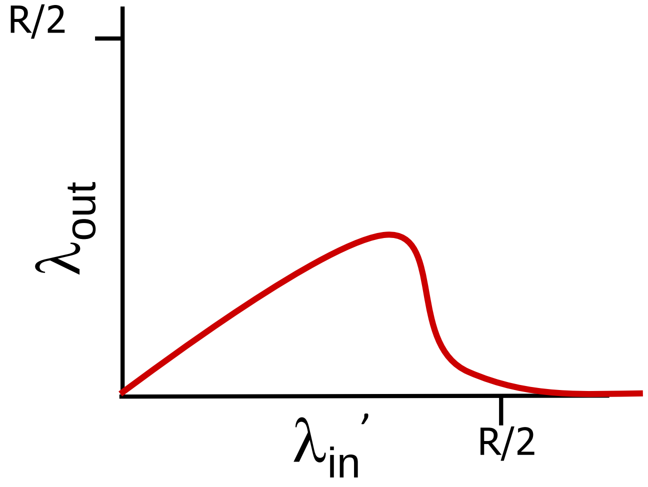
- *four* senders
- *multi-hop* paths
- timeout/retransmit

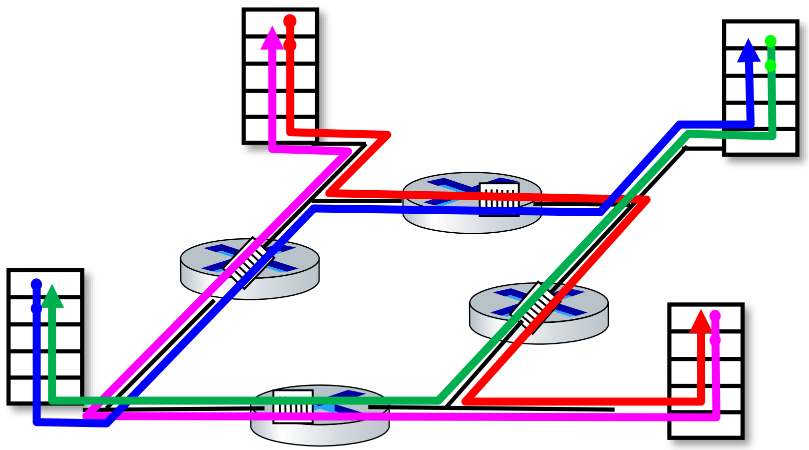
Q: what happens as λ_{in} and λ_{in}' increase ?

A: as red λ_{in}' increases, all arriving blue pkts at upper queue are dropped, blue throughput → 0



Causes/costs of congestion: scenario 3



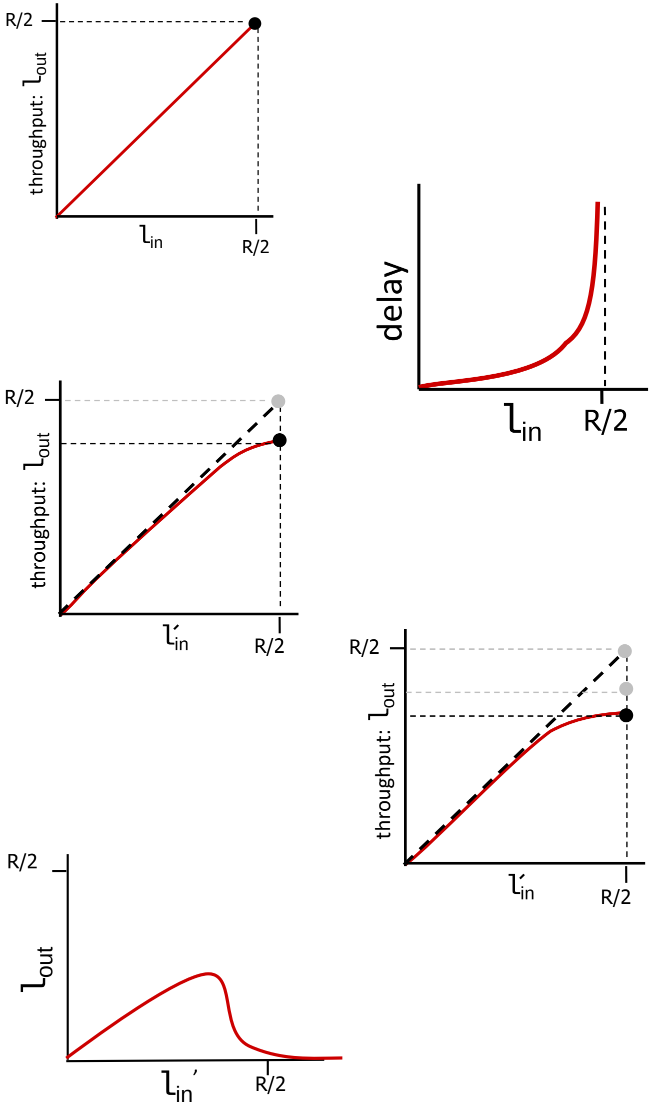


another "cost" of congestion:

- when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted!

Causes/costs of congestion: insights

- throughput can never exceed capacity
- delay increases as capacity approached
- loss/retransmission decreases effective throughput
- un-needed duplicates further decreases effective throughput
- upstream transmission capacity / buffering wasted for packets lost downstream

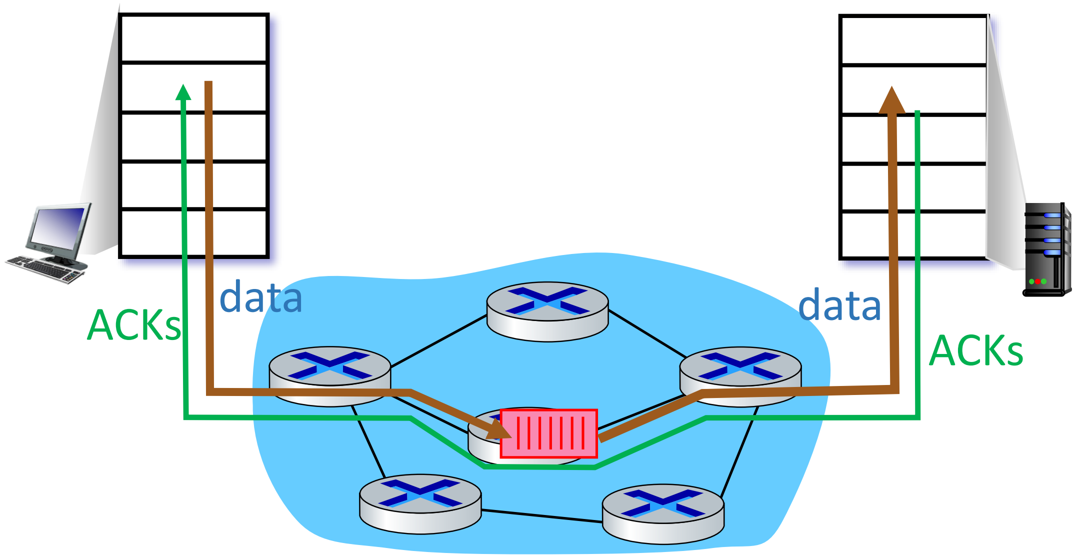


Approaches towards congestion control

End-end congestion control:

- no explicit feedback from network
- congestion *inferred* from observed loss, delay

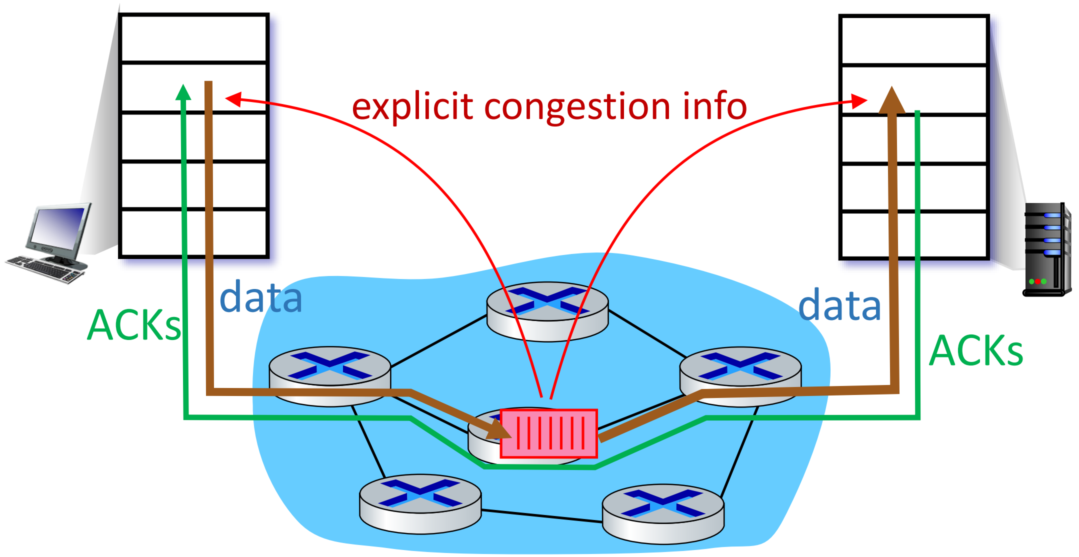
- approach taken by TCP



Approaches towards congestion control

Network-assisted congestion control:

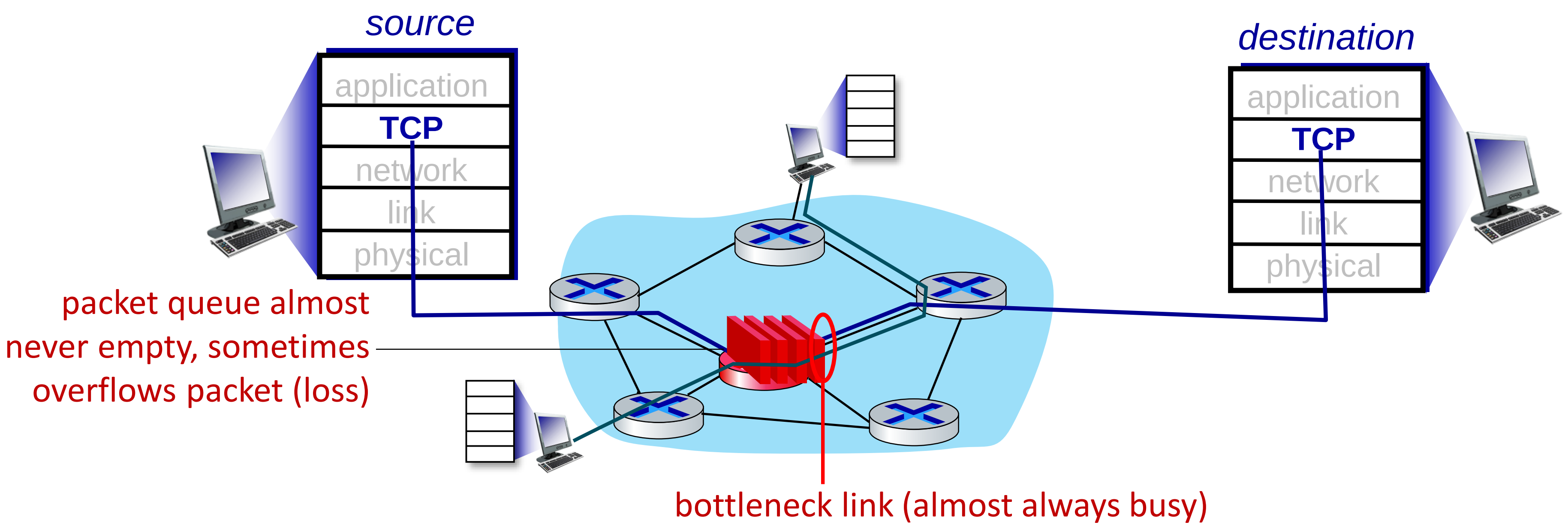
- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate



- TCP ECN, ATM, DECbit protocols

TCP and the congested "bottleneck link"

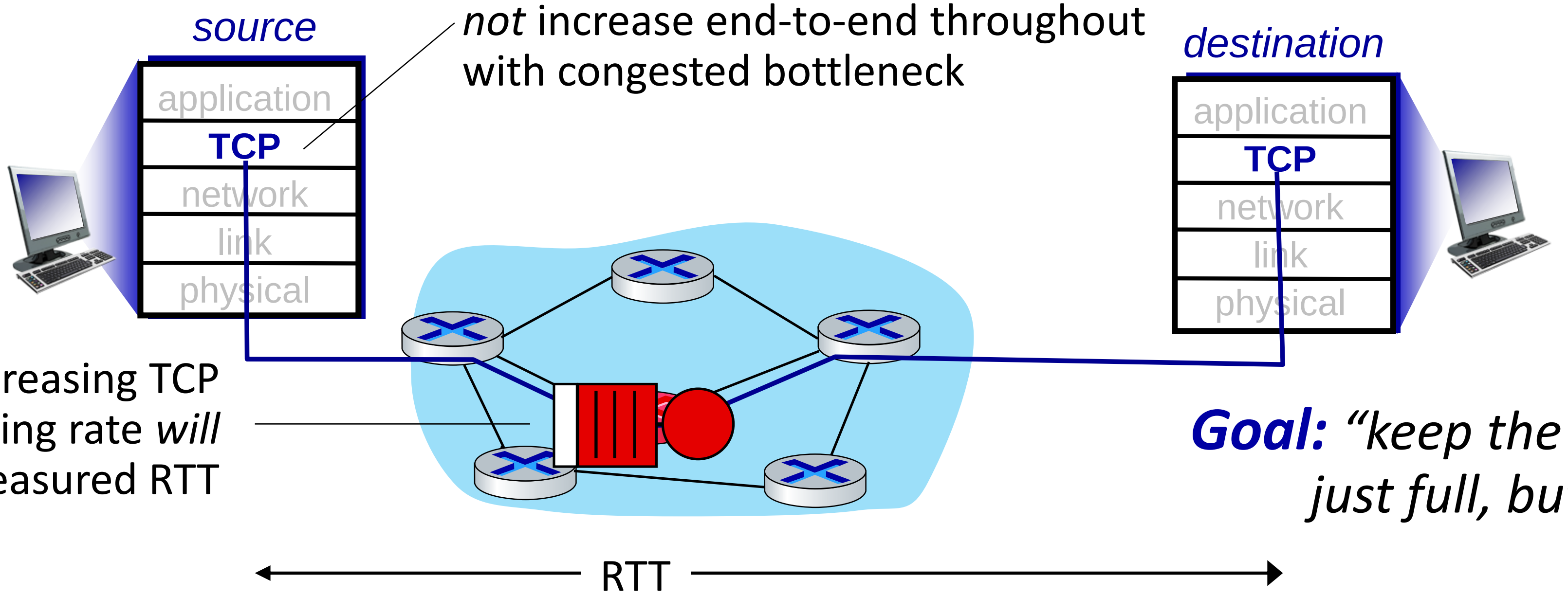
- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*



TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link

insight: increasing TCP sending rate will *not* increase end-to-end throughout with congested bottleneck

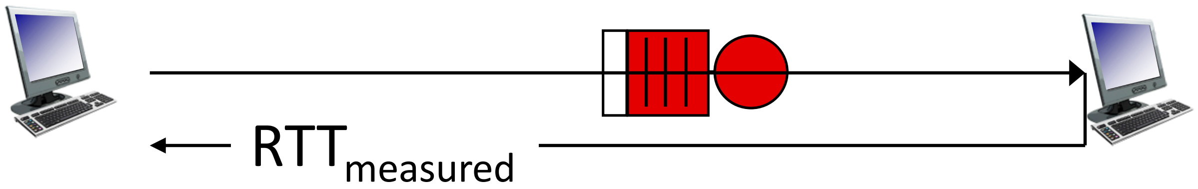


insight: increasing TCP sending rate *will* increase measured RTT

Goal: *"keep the end-to-end pipe just full, but not fuller"*

Delay-based TCP congestion control

Keeping sender-to-receiver pipe "just full enough, but no fuller": keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Delay-based approach:

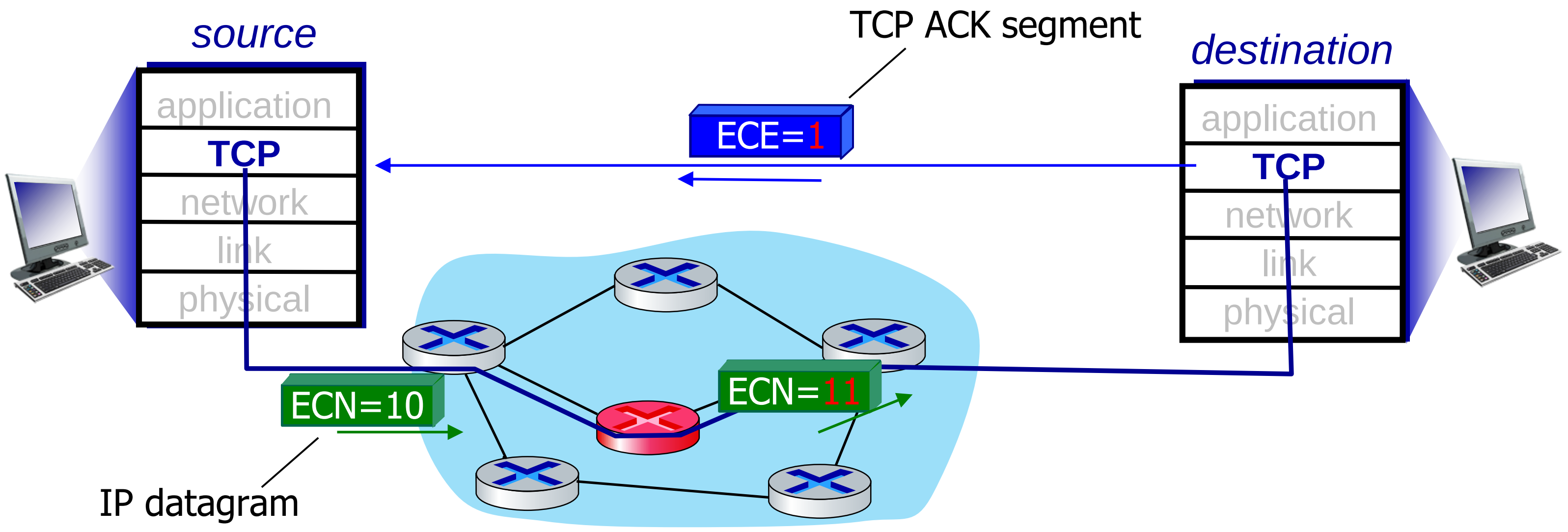
- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is cwnd/RTT_{min}

```
if measured throughput "very close" to  uncongested throughput
      increase cwnd linearly              /* since path not congested */
else if measured throughput "far below" uncongested throughout
      decrease cwnd  linearly             /* since path is congested */
```

Explicit congestion notification (ECN)

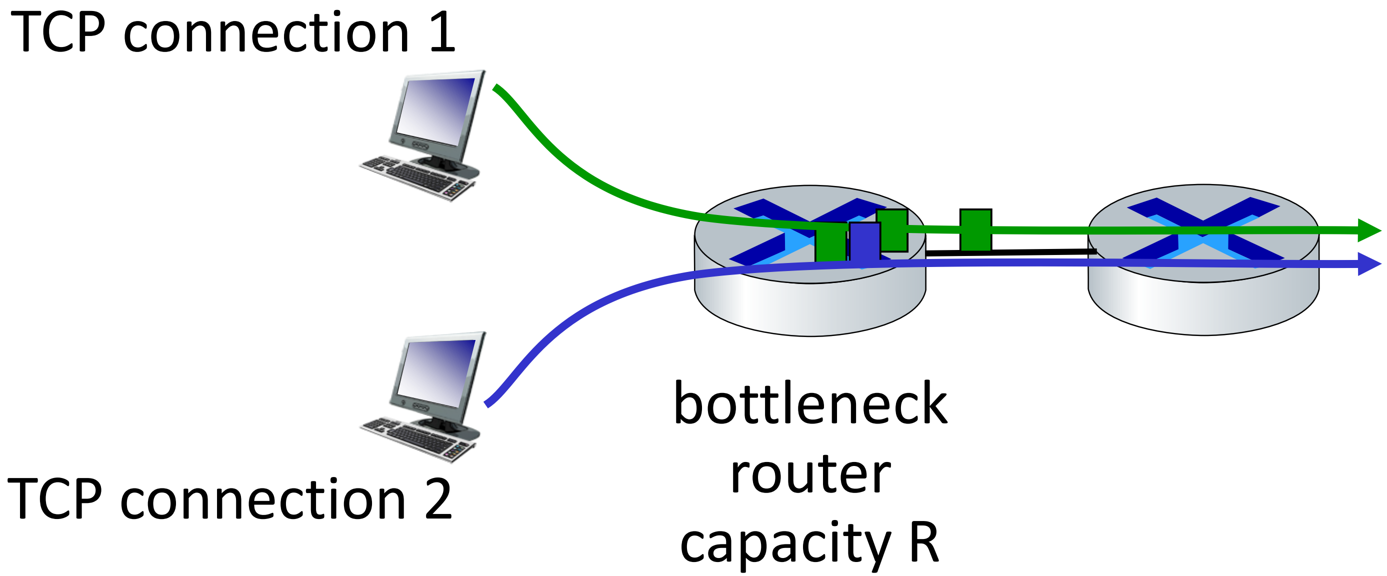
TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



TCP fairness

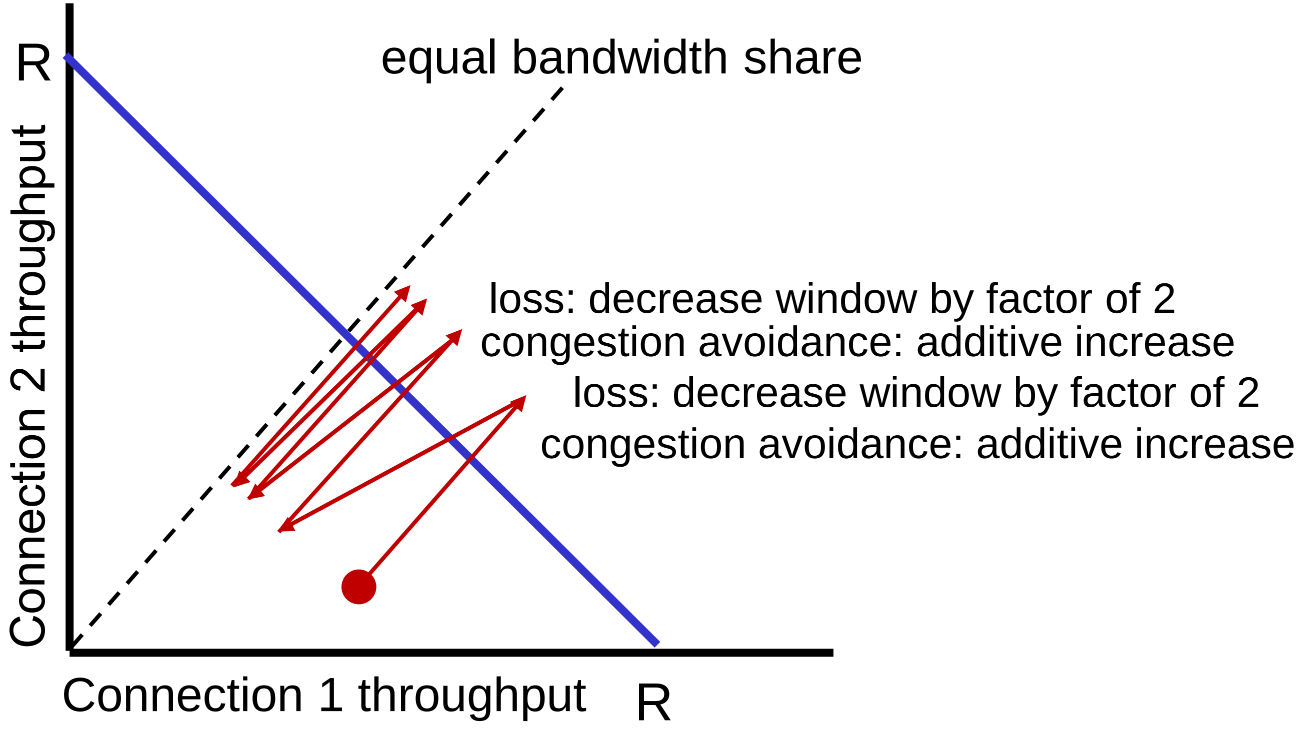
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Fairness: must all network apps be "fair"?

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss
- there is no "Internet police" policing use of congestion control

Fairness, parallel TCP connections

- application can open *multiple* parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10

Network Performance Evaluation

Learning Outcomes

- Understand the importance of simulation
- Analyzing Network Performance
  - Delay
  - Throughput
  - Packets drop/delivery

Roadmap

- Performance Evaluation Approaches
  - Modelling, Simulation, and Deployment
- NS-2
  - Architecture, Trace files and tools
- Otcl scripts for NS2
  - Tcl/Otcl programming language
- Otcl objects and declarations
  - Scheduler, Topology, Transport, Traffic, and Routing
- Simple example
- Appendix:
  - Automate the simulations
  - Random number generations (self-study)

Performance Evaluation Approaches (1/1)

- Three methods used in analysis and performance evaluation
  1. Analytical modeling and solution (costs -)
  2. Simulation of system model(s)
  3. Experimental (costs +)

1. Analytical modeling and resolution (costs -)
   - Closed-form Expression
     - Possible for small scale systems, usually with simplifying assumptions
   - Numerical
     - More complex systems
     - Problems: Computational time, numerical stability

Performance Evaluation Approaches (1/2)

2. Simulation of system model(s)
   - More general
     - The system models hides the complexity of the system
     - Problems: Simulation time - what is the accuracy of the results? Precision?

3. Experimental (costs +)
   - Deployment of real test-beds
     - Problems: Scalability, Repeatability-what is the accuracy of the results? Precision?

Network Simulator: Version-2 (NS-2)

- NS2 is a discrete-event simulator for networking
- Simulations of events related to packets
- Simulations of PHY/MAC layer to application layer
  - Through PHY, MAC,…, and Application models

- Wired, Wireless, Mesh, and Satellite networks
- Examples of simulated protocols:
  - MAC: CSMA/CD, CSMA/CA, 802.11,BlueTooth, GPRS
  - Network: IP, unicast and multicast routing, mobile IP, …
  - Transport: UDP, TCP Reno, TCP Tahoe, SACK,…
  - Application: HTTP, VoIP, etc, ... (more precisely traffic generation)

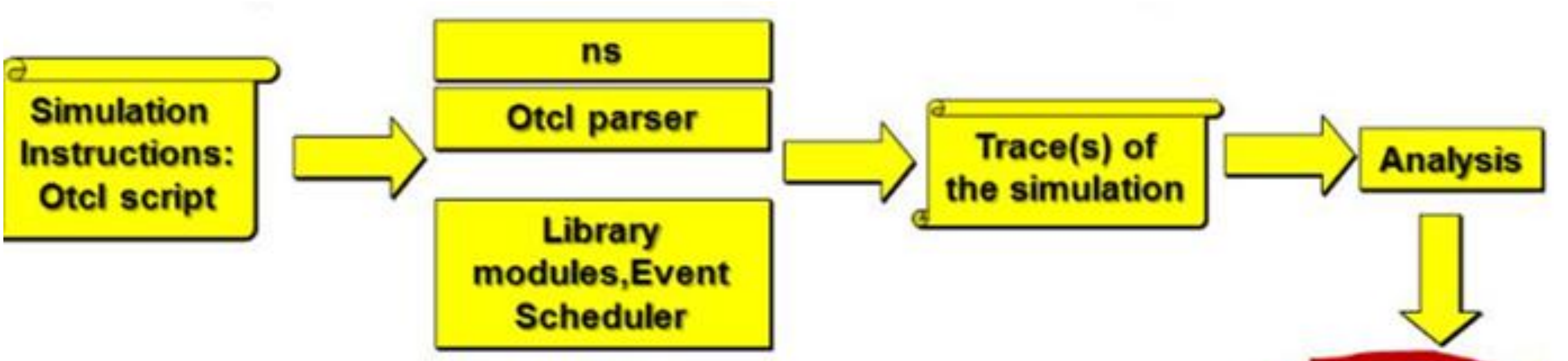
NS-2: Architecture

- Object oriented: *C++* and *Otcl*
- C++: Implements the simulation of events:
  - The event scheduler
  - Network modules
  - Packed “movement”
- C++: fast execution

- Otcl: implements user interface: simulations control:
  - Simulation scripts
  - Configuration of simulations parameters
  - Link to C++ modules
- Otcl: Simple, scripts are commands to the simulator.

Simulation Steps

- From a user perspective
  - Write an OTCL script in order to
    - Initialize the event scheduler
    - Define the simulation scenario and parameters, e.g. simu.tcl



- Run the simulation with the command **ns simu.tcl**
- Analyze the executed events to get the results

Traces and tools: Simple network trace files

- Trace format

event | *time* | from | *to* | pkttype | *pktsize* | flags | *fid* | src | *dst* | seq | *pktid*

Example

```
r 1.3556   3 2 ack 40   ------- 1 3.0 0.0 15  201
+ 1.3556   2 0 ack 40   ------- 1 3.0 0.0 15  201
- 1.3556   2 0 ack 40   ------- 1 3.0 0.0 15  201
r 1.35576  0 2 tcp 1000 ------- 1 0.0 3.0 29  199
+ 1.35576  2 3 tcp 1000 ------- 1 0.0 3.0 29  199
d 1.35576  2 3 tcp 1000 ------- 1 0.0 3.0 29  199
+ 1.356    1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356    1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

r: reception, +: arrival in the queue, -: departure from the queue, d: drop

IP address = Node_id.port_number (no need for real IP addresses or port numbers)

Network interface of a router = queue

Traces and tools: Trace analysis(1/3)

- Example: Awk
  - Language suitable to parse text files.
- Easy to learn, and widespread, e.g. http://www.vectorsite.net/tsawk.html
  - Easy to user and to run
- The main frame of the program

```
BEGIN { <initialization>}
  {<program core (parsing instructions)>}
END {<final actions>}
```

  - Run from the shell: awk –f program.awk tracefile
- Of course any other language can be used to parse trace files
  - e.g. Perl, C, Python , Java

Traces and tools: Trace analysis(2/3)

- Example

```
BEGIN { maxid = 0; nbp = 0; }
```

Initialization

```
{ action = $1;
time = $2;
node0 = $9;
node1 = $10;
pid = $12;

if(pid > maxid) maxid = pid;
if(action == "+" && node0 == 0)
        sendingTime[pid] = time;
if(action == "r" && node1 == 1) {
        received[pid] = 1;
        receivedTime[pid] = time;}
}
```

This code is applied for each line of the input file

The fields of each line are obtained straightforwardly by $i, i is the i^th field

Traces and files (3/3)

- Example Continued …

Code applied after parsing the whole file (computations, printing, checking, ...)

```
END {
for(pid = 0; pid <= maxid; pid++) {
        if(received[pid] == 1) {
                nbp++;
                delays += receivedTime[pid]-sendingTime[pid];
        }
}
printf("average delay = %f\n", 1.0 * delays/nbp);
}
```

Traces and tools: xgraph, gnuplot

- Xgraph: plot quickly graphs

```
% xgraph "file"
% xgraph -help
```

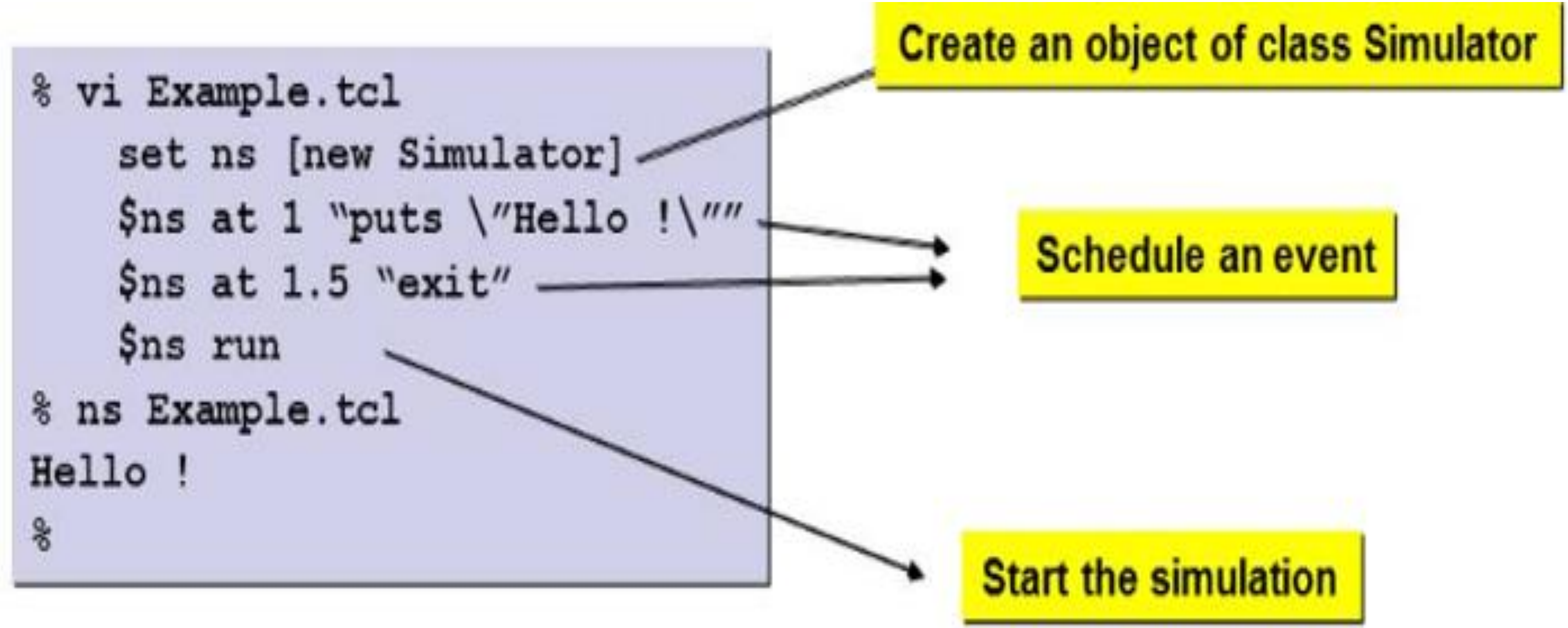
- 'file' includes two fields(x,y) per line.
- Gnuplot: same but with more options and graphical configurations

```
%gnuplot
gnuplot> set output "image.jpg"
gnuplot> set terminal jpeg
gnuplot> plot "file" using 1:2 with lines
```

- "file" includes several fields per line
- Of course others: Matlab, Maple, …

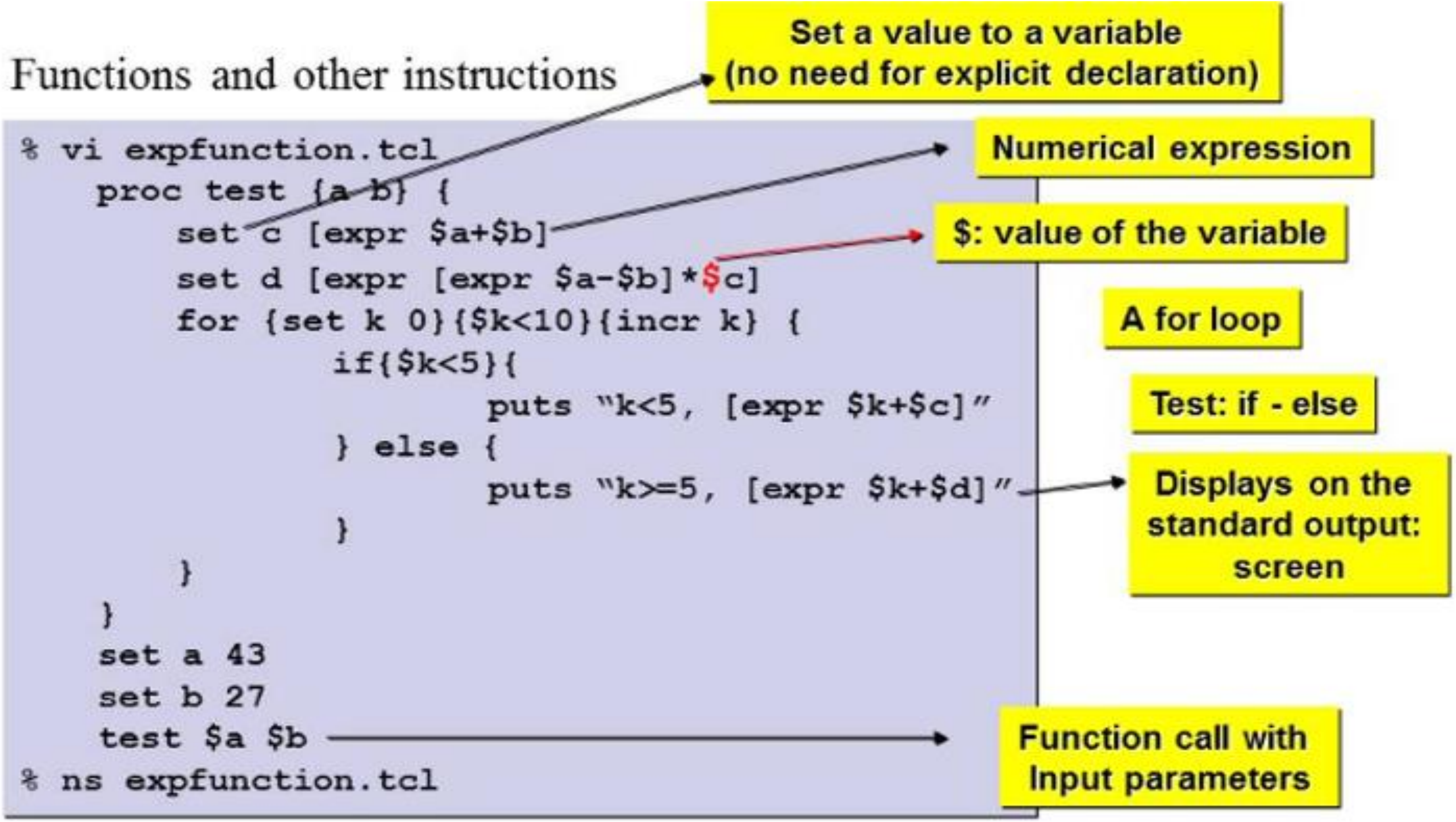
Otcl Scripts: Tcl/Otcl programming(1/3)

- Simple example of a tcl script for ns2



Otcl Scripts: Tcl/Otcl programming(2/3)

- Functions and other instructions

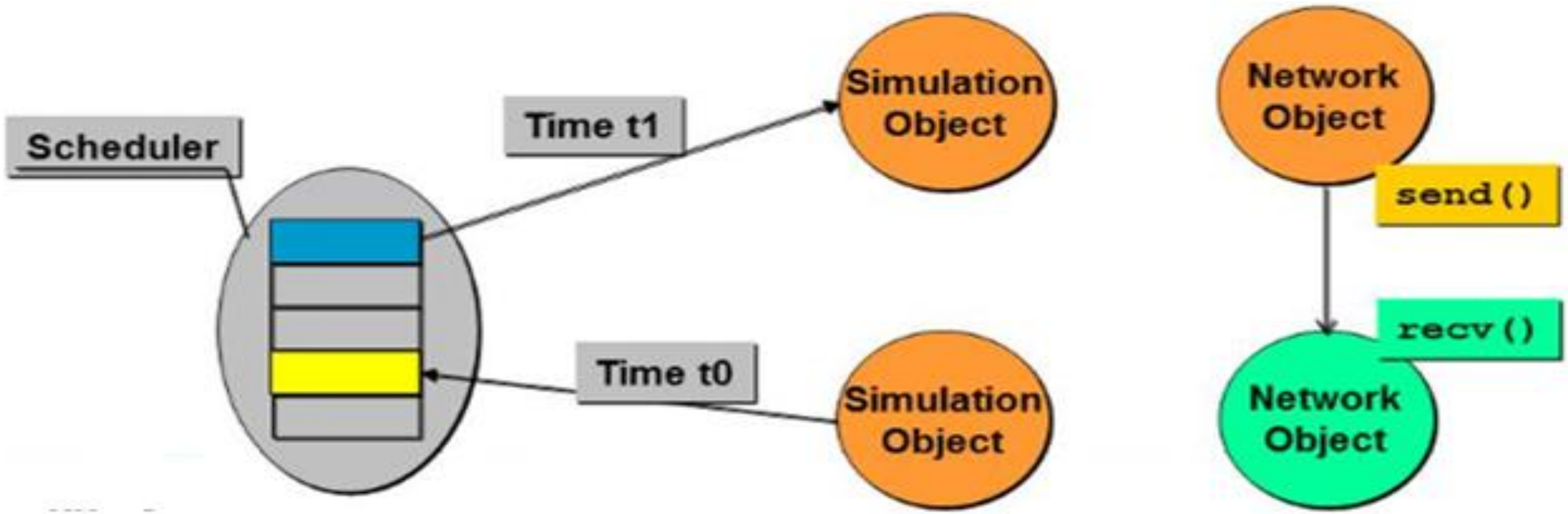


Otcl scripts: Typical content and steps for ns2 simulations

1. Create an object of class simulator (includes the event scheduler)
2. Create and/or open trace file(s)
   - tr and name
3. Define the network topology
   - nodes, links, mobility, loss modules, queues, bandwidths, propagation delays etc.
4. Define the traffic scenario
   - Transport protocols, routing, traffic generators (type, packet size, starting time, end time etc.)
5. Set values to some attributes/variables used in the simulation
6. Set the simulation time
8. Add possibly other events
9. Start the simulation

The scheduler(1/2)

- The scheduler computes and handles events
- Most of events are related to packets
- Packets move from one module to another without any delay
- Transmission delays are computed to determine times of events



The scheduler(2/2): OTCL

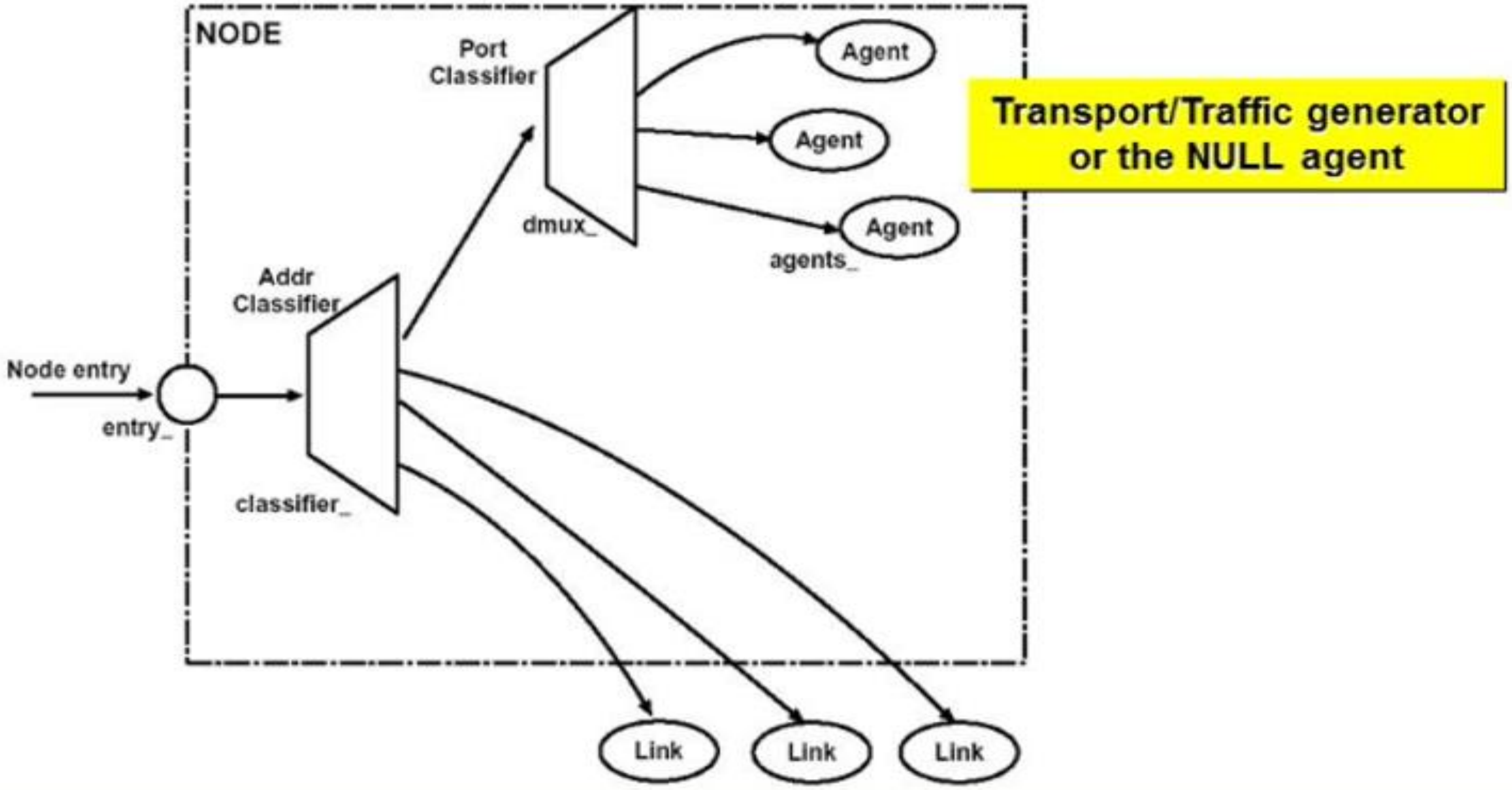
- Create an object of class scheduler: set ns [new simulator]
- Get current simulated time: $ns now
- Stop the simulation: $ns halt
- Start the simulation: $ns run
- Scheduler an event (task): $ns at <time><event>
- Check if the simulation is running: $ns is-started
- Set the type of the scheduler: $ns use-scheduler <type>
  - Type=List|Calendar|Heap|RealTime
  - By default, Type=Calendar

Nodes (1/3)

- Attributes of a node:
  - A unique identifier id_
  - A list of its neighbors neighbor_
  - A list of associated agents agent_
  - A type nodetype_
  - A routing module
  - An entry point entry_ for incoming packets

Nodes (2/3)

- Structure of a node (unicast)

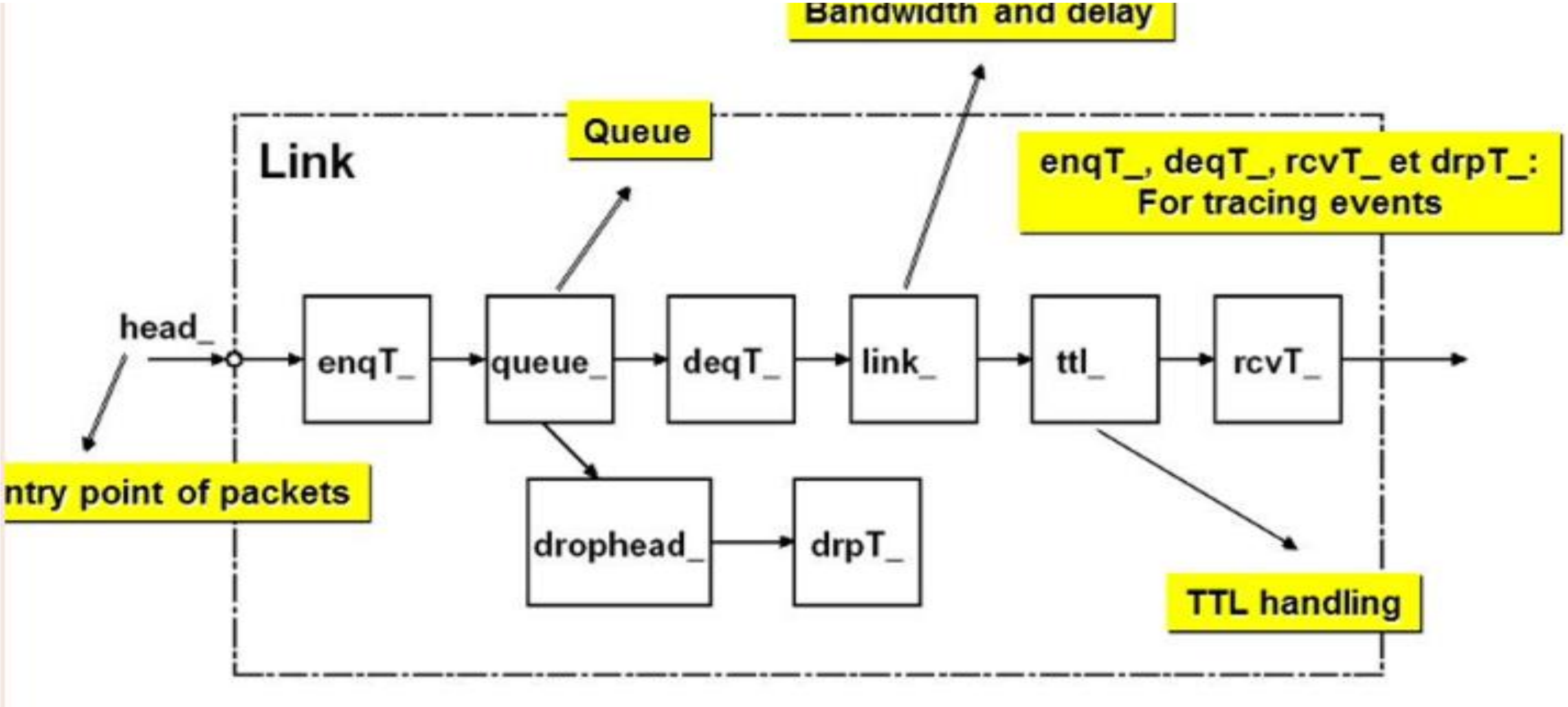


Nodes (3/3): OTCL

- Create a node: set nd [$ns node]
- Enable multicast nodes:
  - set ns [new simulator –multicast on]
- Get the node id: $nd id
- Get the address of the node: $nd node-addr
- Get the entry point: $nd entry
- Get neighbors of the node: $nd neighbors
- Get an agent $nd agent <port>
- Reset a node $nd reset

Link (1/2)

- Structure of a link



Links (2/2) OTCL

- Create a one-way link:

$ns simplex-link <node1> <node2> <bandwidth> <delay> <Type of the queue> <opt:arguments>

- Create a duplex link: $ns duplex-link
- Get the link that connects two nodes: node1 node2

  set link12 [$ns link <node1> <node2>

- Get head_: $lnk head
- Get link_: $lnk link
- Get queue_: $lnk queue

Queues

- Available types of queues:
  - Queue/DropTail (DropTail+FIFO)
  - Queue/RED (RED+FIFO)
  - CBQ (DT+CBQ)
  - FQ (Fair Queuing)
  - SFQ (Stochastic Fair Queuing)
  - DRR
  - …
- Limit the queue size in packets:

  $ns queue-limit <node1> <node2> <limit>

Loss/Error modules

- Instantiate an error module: set losses [new ErrorModel]
- Set the unit and error ratio:

  $losses unit <pkt|byte|time>

  $losses set rate_ <ratio>
- Instantiate the random variable: $losses ranvar [new RandomVariable/Uniform]
- Destination of lost packets:
  - $losses drop-target [new Agent/Null]
- Insert the error module between two nodes:
  - $ns lossmodel <module> <noeud1> <noeud2>
- Example: $ns lossmodel $losses $n1 $n2

Transport Protocols: Agents(1/2)

- UDP sender: Agent/UDP
- TCP sender
  - Tahoe: Agent/TCP
  - Reno: Agent/TCP/Reno
  - New Reno: Agent/TCP/Newreno
  - Selective repeat: Agent/TCP/Sack1
  - Vegas: Agent/TCP/Vegas
  - Forward Ack: Agent/TCP/Fack
- TCP receiver:
  - Basic one: Agent/TCPSink
  - Delayed Ack: Agent/TCPSink/DelAck
  - Selective Acknowledgment: Agent/TCPSink/Sack1

Transport Protocols: Agents(2/2)

> Agent UDP: Sender

```
set udp [new Agent/UDP]
$ns attach-agent <node1> $udp
$udp set packetSize_ <packet size in bytes>
$udp set fid_ <flow id>
```

Instantiate the agent

Add the agent to the list of node's agents

Set the packet size

Set a flow id that will be mentionned in the trace file

> Receiver

```
set null [new Agent/Null]
$ns attach-agent <node2> $null
```

No actions at the UDP receiver

> Connection

```
$ns connect $udp $null
```

Create a transport-level end-to-end connection between the two modules

Traffic Generators

- Principle most used one:
  - Application/Traffic/Exponential: ON/OFF Exponential
    - Variables: rate_, burst_time_, idle_time_, packetSize_
  - Application/Traffic/CBR: constant bit rat
    - Variables: rate_, burst_time_, idle_time, packetSize_, shape_
  - Application/Traffic/FTP: file transfer protocol

- Procedures:
  - attach-agent: Associate to an agent
  - use-rng: Use a particular random number generator.
  - start: Start generating traffic
  - stop: Stop generating the traffic

A simple example: Two nodes + one link (wired network)

- Scenario

A simple example: Two nodes + one link (wired network)

- Create the scheduler and open a trace file

```
set ns [new simulator]
$ns trace-all [open test.out w]
```

- Create the topology

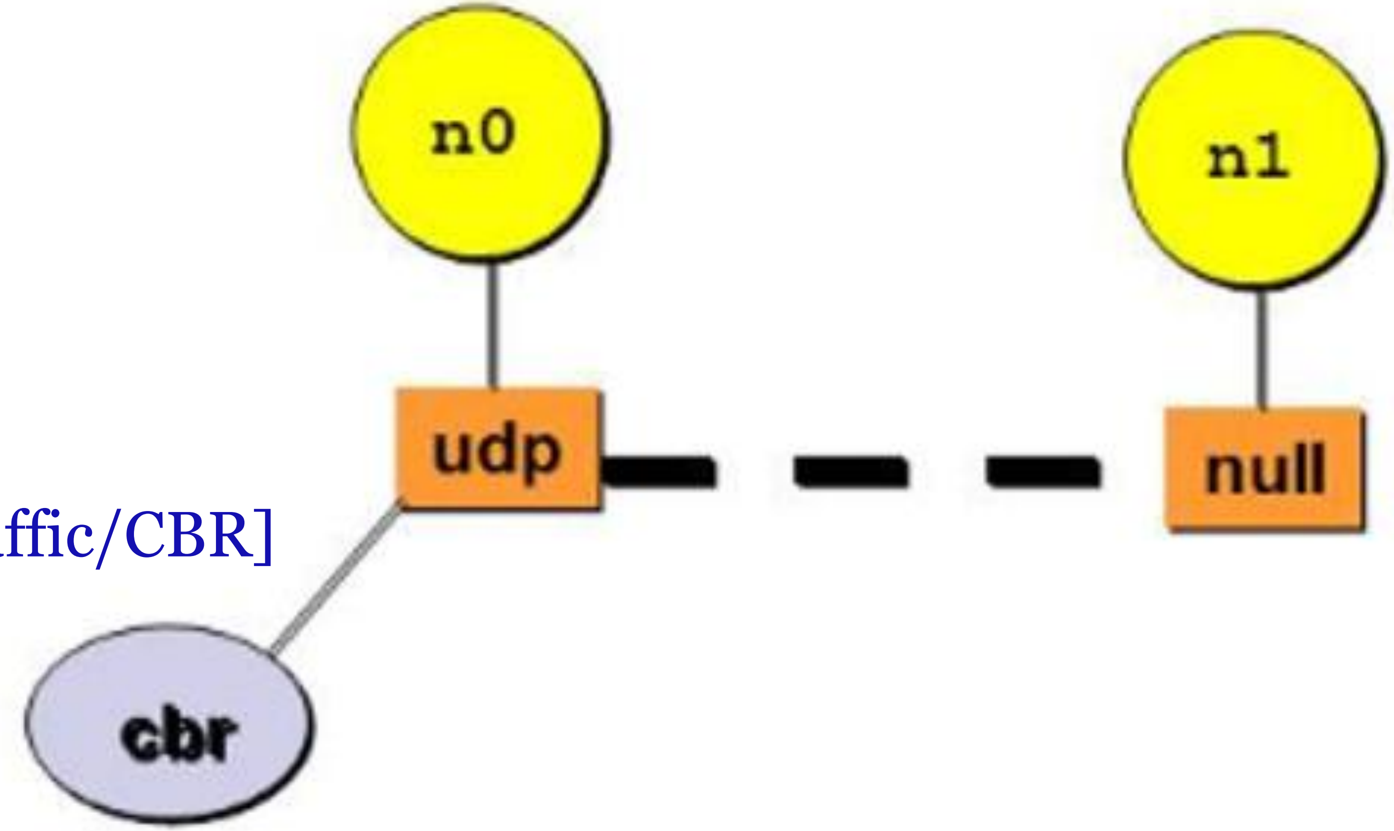
```
set n(0) [$ns node]
set n(1) [$ns node]
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail
$ns queue-limit $n(0) $n(1) 10
```

A simple example: Two nodes + one link (wired network)

- UDP and CBR traffic generator

```
set udp [new Agent/UDP]
$udp set fid_ 2
$udp set packetSize_ 1024
$ns attack-agent $n(0) $udp
set null [new Agent/Null]
$ns attack-agent $n(1) $null
$ns connect $udp $null

Set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetSize_ 1024
$cbr set rate_ 512kb
```



A simple example: Two nodes + one link (wired network)

- Start and end events of traffic generation

  $ns at 0.1 "$cbr start"

  $ns at 4.5 "$cbr stop"

- Set the event of the end of the simulation and start the simulation

  $ns at 5.0 "$ns halt"

  $ns run

Appendix I: Automate the simulations(1/3)

- Example of a shell script
- Run the same simulation code with different input parameters

```
#! /bin/sh -fx
mesSim = "1 2 3 4 5 6 7 8 9 10"
mesChargesdentree = "50 100 150 200"

for i in $mesSim
do { for rt in $mesChargesdentree
     do { ns fichier.tcl $rt $i out.tr
          awk -f prog.awk out.tr >> resultats.out
     } done
} done
```

- Get the input parameters inside the tcl code

```
set lambda [lindex $argv 0]
set seed [lindex $argv 1]
set trcafile [lindex $argv 2]
```

Appendix I: Automate the simulations(2/3)

- Example of a shell script (for advanced users)
- Using "- OptionName" before each input parameter

```
#! /bin/sh -fx
mesSim = "1 2 3 4 5 6 7 8 9 10"
mesChargesdentree = "50 100 150 200"

for i in $mesSim
do { for rt in $mesChargesdentree
     do { ns fichier.tcl -rate $rt -seed $i -tr out.tr
          awk -f prog.awk out.tr >> resultats.out
     } done
} done
```

Appendix I: Automate the simulations(3/3)

- Get the input parameter that follow all "OptionName"(for advanced users)

```
…
proc getopt{argc argv} {
        global opt
        lapend optlist rate seed tr
        for{set i 0}{$i < $argc}{incr i} {
                set arg [lindex $argv $i]
                if{[string range $arg 0 0] != "-"} continue
                set name [string range $arg 1 end]
                set opt($name) [lindex $argv [expr $i+1]]
        }
}
…
set tracefd [open $opt(tr) w]
…
ns-random $opt(seed)
…
$cbr set rate_ $opt(rate)
```

Appendix: Generation of random numbers

- Previously:
  - We use physical phenomenon such as noise and radio activity emission, dice, etc.
- Now:
  - We use computational algorithms that generates pseudo-random numbers.
- Some methods:
  - Congruential
  - Register shifting
  - Brewing
- Brewing creates a random number generator from several different generators

Appendix: Generation of random numbers

- Example: A mixed congruential method
  - Random numbers are generated through a numerical sequence
  - $U_k = aU_{k-1} + b \ (mod\ m)$
- With a = 3141592654
  b= 453806245
  $m = 2^{31}$
  - U_0: is the seed of the generator
  - m: congruent factor
  - a: multiplier
  - b: additive factor
- Changing the seed U_0 provides a different sequence of random numbers

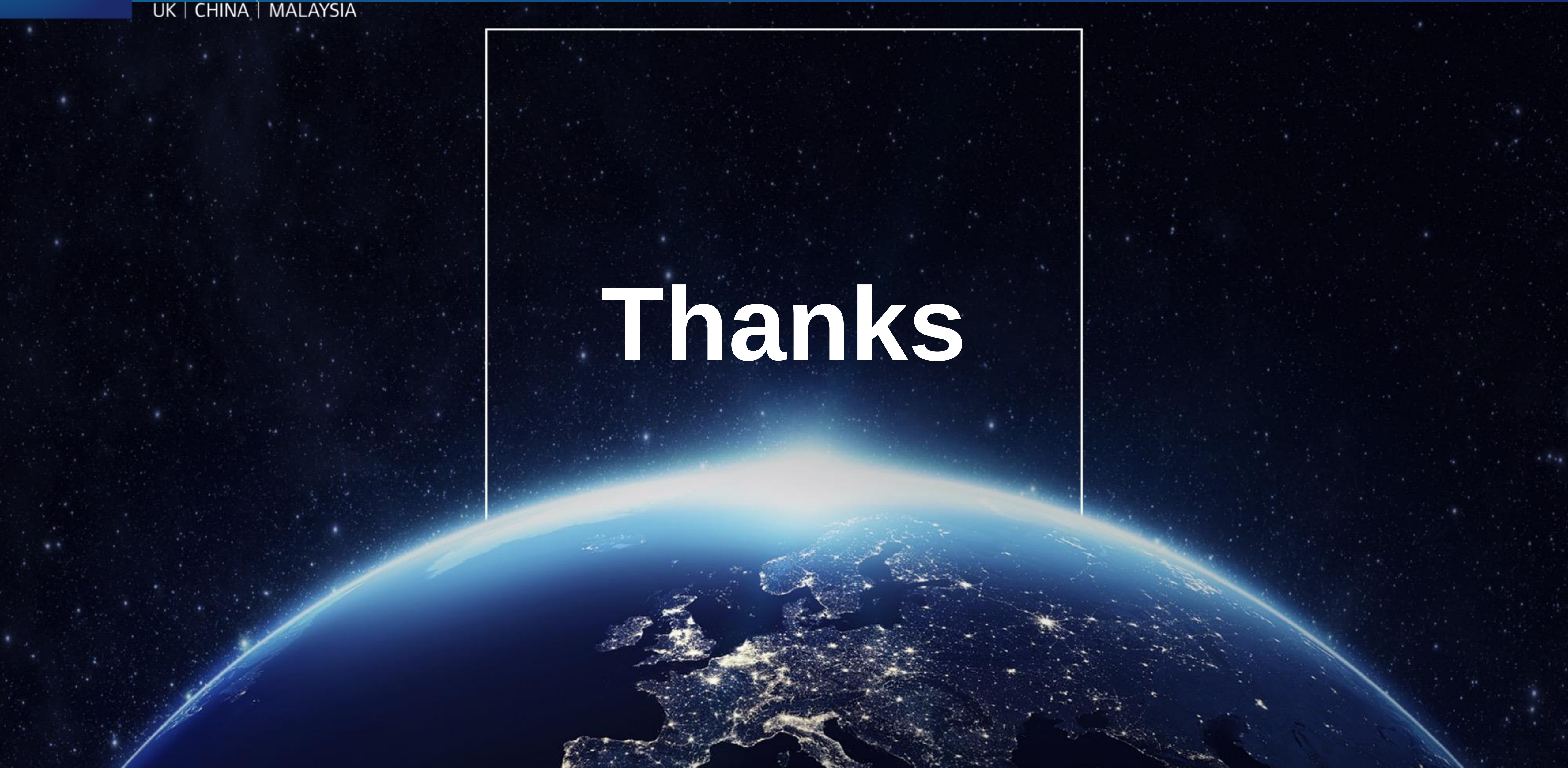
Appendix: Generation of random numbers

- Programming languages provides ready-to-use functions to initialize the seed and generate random numbers.
- Most simulators provides modules that generate a variety of distributions
  - Exponential
  - Pareto
  - Normal
  - Etc.
- NS-2: Several distributions including the exponential ON/OFF

References

- Marceau Coupechoux. “Introduction a NS2”
- Kevin Fall, Kannan Varadhan. “The ns Manual”. VINT project UC Berkeley, USC/ISI, and Xerox PARC.

University of Nottingham
UK | CHINA | MALAYSIA



Thanks