Modeling of data networks by example: NS-2 (II)

Holger Füßler

Course overview

1. Introduction

2. Building block: RNG

3. Building block: Generating random variates I and modeling examples

4. Building block: Generating random variates II and modeling examples

5. Algorithmics: Management of events

6. NS-2: Introduction

7. NS-2: Fixed networks

8. NS-2: Wireless networks

9. Output analysis: single system

10. Output analysis: comparing different configurations

11. Omnet++ / OPNET

12. Simulation lifecycle, summary

Lecture overview

- Retrace and understand a typical use case for simulation of computer networks
  - Compare two different TCP congestion control variants (TCP Tahoe and TCP Reno)
  - Create a model and set up an ns-2 simulation script
  - Run the script
  - Have a look at the result (the files / with nam)
  - Create a simple parser script and produce ‘semantic-rich’ output

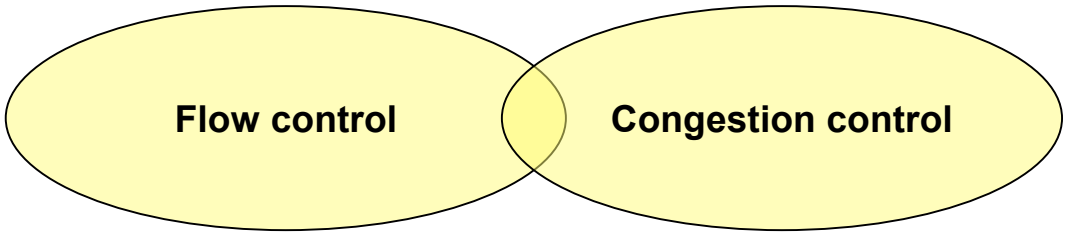
- Part I: Basics on TCP
- Part II: Ns-2 scripts for TCP simulations
- Part III: Regular expressions for output analysis
- Part IV: Results and wrap-up

I Problem statement

- Assume we have TCP Tahoe RFC and widely installed
- Van Jacobson: “I want to improve TCPs responsiveness to packet reordering”
- First step: understand how it works and what goes wrong / could be optimized
  - read papers about algorithms
  - set up and analyze example with ns-2
- Second step: program (potential) improvement
  - tune algorithm
  - compare algorithms

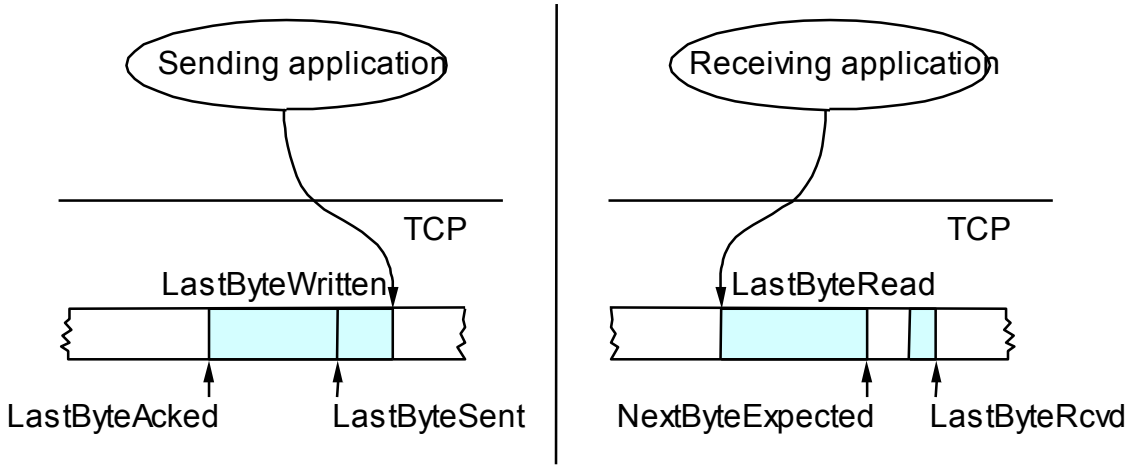
I Recap: Basics on TCP

- TCP guarantees reliable, in-order delivery, connection-oriented, in-order byte-stream service
- … full-duplex
- … has flow and congestion control mechanisms included:



- Flow control: involves preventing senders from overrunning the capacity of receivers, reliable transport
- Congestion control: involves preventing too much data from being injected into the network

I Recap: TCP flow control (sliding window)



» Sending side

- LastByteAcked < = LastByteSent
- LastByteSent < = LastByteWritten
- **buffer bytes between** LastByteAcked **and** LastByteWritten

» Receiving side

- LastByteRead < NextByteExpected
- NextByteExpected < = LastByteRcvd +1
- **buffer bytes between** NextByteRead **and** LastByteRcvd

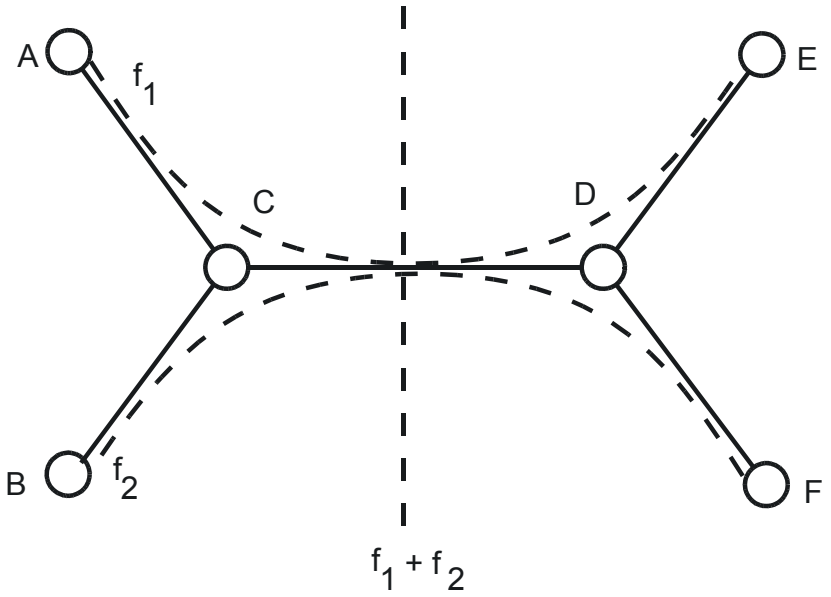
[Peterson/Davie: Computer Networks]

I Recap: TCP retransmission timeout

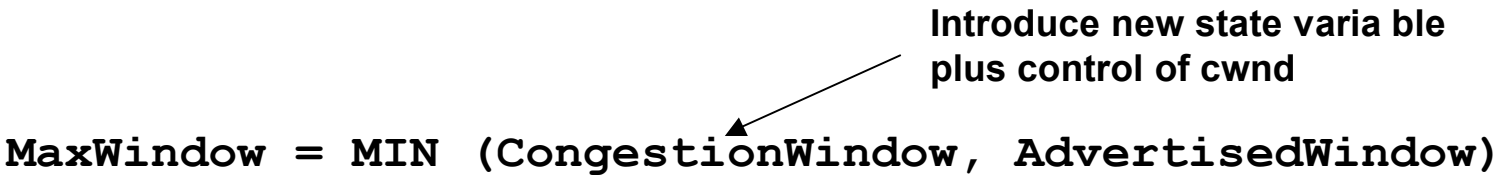
- Adaptive retransmissions
- Timeout value computed via estimate of RTT
- Karn/Partridge:
  - Solves original flaw
  - Includes exponential backoff
- Jacobson/Karels:
  - Includes estimation of variance of RTT

I TCP Congestion Control (Basics)

- IP is best-effort datagram
- Several Flows compete for „bottleneck link“
- How does it work?
  - routers drop IP packets on congestion
  - end-systems detect this packet loss via time-out
  - end-systems „voluntarily“ reduce rate



I TCP Congestion Control (Basic)



» TCP Tahoe (4.3 BSD Tahoe)

- slow-start (cwnd < ssthresh)

  senders starts with cwnd of 1 segment, exponentially increasing until ssthresh is reached or loss is detected
- congestion avoidance (cwnd > ssthresh)

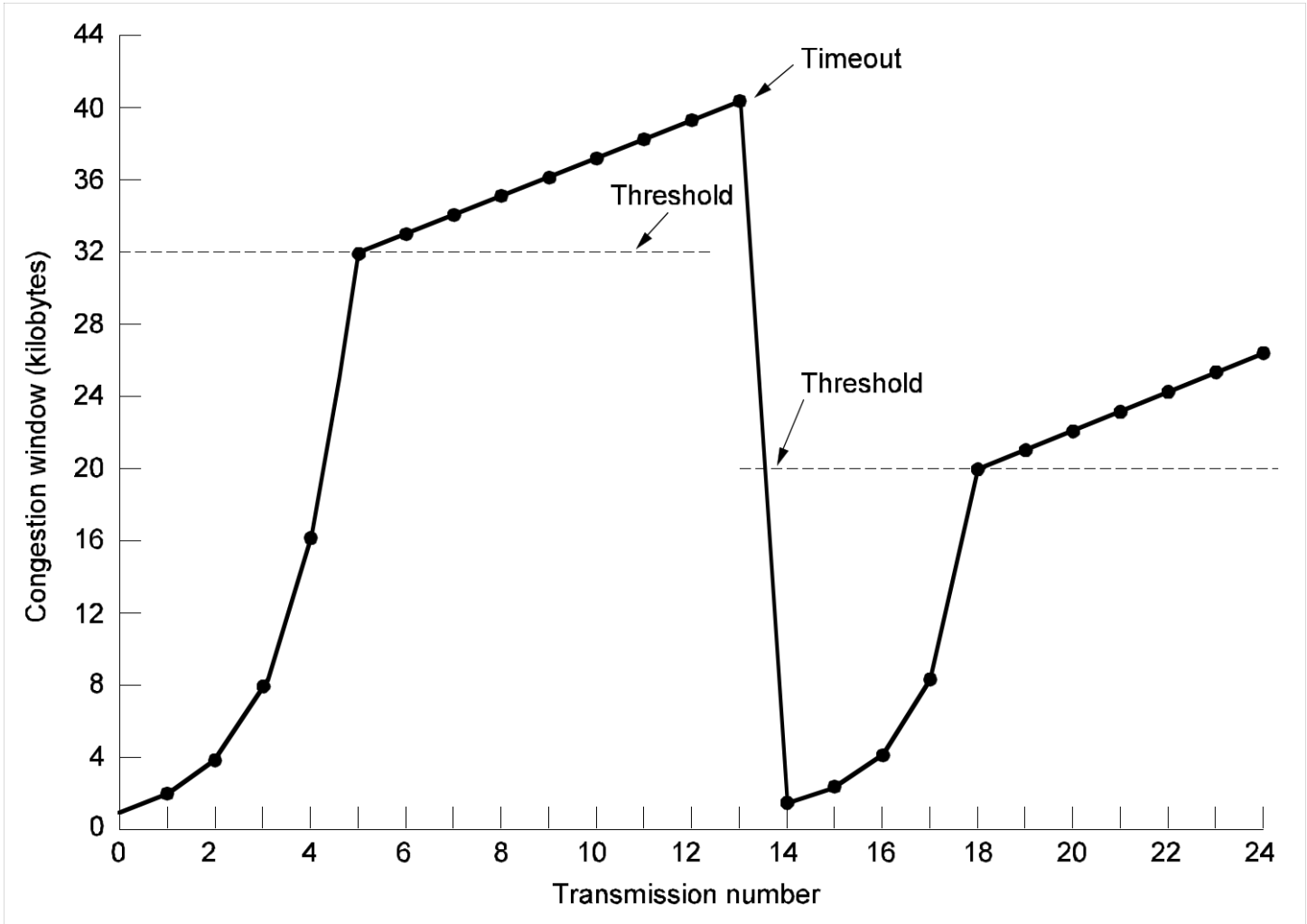
  senders increases cwnd linearly until loss is detected
- fast retransmit

  after receiving 'triple duplicate ack' (TDACK) missing segment is retransmit without waiting for the request to time out

cwnd is congestion window size

ssthresh is slow start threshold

I TCP Tahoe congestion control example



[Tanenbaum, CN3]

I TCP Congestion Control (Improvement)

- In Tahoe, both RTO and TDACK result in a cwnd of 1 and the initiation of the “slow-start” algorithm
- BUT: What if the network has a high bandwidth-delay product (b-d p)?

→ Overall throughput will be low!

→ TDACK means that “one / some segments” are missing, RTO means “all segments are missing”

→ idea: only set cwnd to ssthresh as opposed to 1 and use “congestion avoidance”

cwnd is congestion window size

ssthresh is slow start threshold

RTO is retransmission timeout

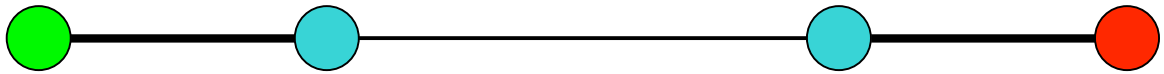
TDACK is „triple duplicate ack“

b-d p is „bandwidth-delay product“

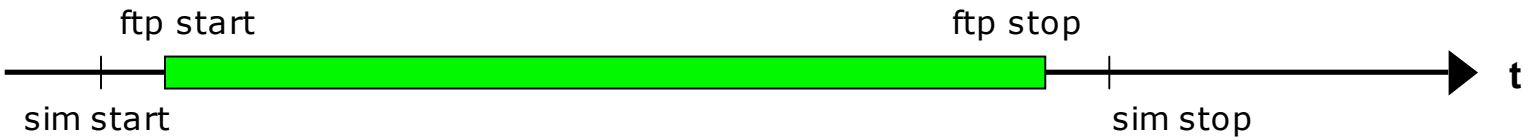
„Fast recovery“ (Reno)

II Simple ns-2 setup for TCP Tahoe

- What do we need?
- (Simple) wired network topology
  - just as complex as it has to be to show problems
  - two hosts two ‘routers’, one ‘bottleneck link’
  - static routing, no errors



- Data communication scenario
  - one host sends data to other host using a single ‘router’
  - hosts sends for a time of some seconds
  - in this time it sends “as much data as the network allows”



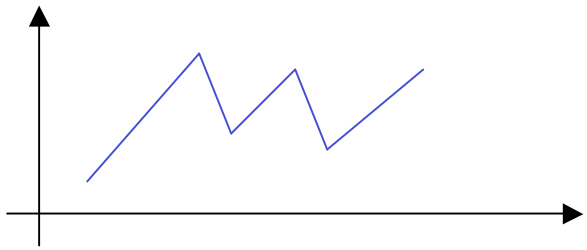
II Plan your results

» **Which parameters are we interested in?**

- → **Total amount of data received at data sink (no packets / no bytes)**
- → **Total number of packet drops**
- → **Size of congestion window over time**

» **What values do we expect?**

- → **upper limit of data received** $D_{\max} = \tau \min_i \{lbw_i\}$
- → **where τ is the time of the data flow and lbw_i are the link bandwidths**
- → **number of packet drops: cannot say, but we expect some (depends also on queue length we set)**
- → **Size of congestion window – expect plot over time**



II Setting up the TCP Tahoe simulation

```
set ns [new Simulator]

# [Turn on tracing]

# Create topology

# Setup packet loss, link dynamics

# Create routing agents

# Create:

# - multicast groups

# - protocol agents

# - application and/or setup traffic sources

# Post-processing procs

# Start simulation
```

II The TCL script (Tahoe)

```
set ns [new Simulator]
```

Create Simulator Object

```
set simtype "tcp-tahoe"
```

Set String Variable

```
$ns trace-all [open ${simtype}.tr w]
$ns namtrace-all [open ${simtype}.nam w]
```

Open File Descriptors (aka TCL Channels)

```
foreach i " 0 1 2 3 " {
        set n$i [$ns node]
}
```

Create Nodes

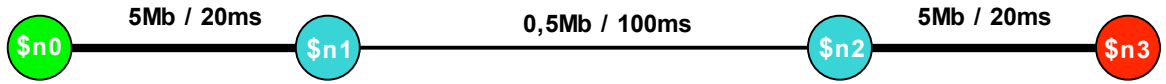
```
$ns at 0.0 "$n0 label TCP"
$ns at 0.0 "$n3 label TCP"
```

Set Node Labels (for nam)

II The TCL script (Tahoe) 2

```
$ns duplex-link $n0 $n1 5Mb 20ms DropTail
$ns duplex-link $n1 $n2 0.5Mb 100ms DropTail
$ns duplex-link $n2 $n3 5Mb 20ms DropTail
```

Create Topology




```
$ns queue-limit $n1 $n2 5
```

Set queue size

```
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient right
```

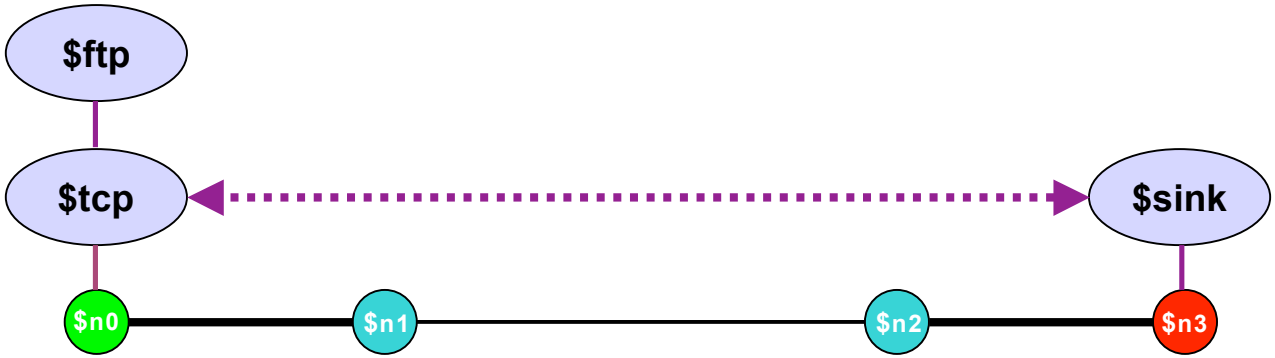
Drawing Hints for nam

```
$ns duplex-link-op $n1 $n2 queuePos 0.5
```

II The TCL script (Tahoe) 3

```
Agent/TCP set nam_tracevar_ true
Agent/TCP set window_ 64
Agent/TCP set ssthresh_ 20

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns add-agent-trace $tcp tcp
$ns monitor-agent-trace $tcp
$tcp tracevar cwnd_
$tcp tracevar ssthresh_
```

Set global options for Agent/TCP



Set trace options

II The TCL script (Tahoe) 4

```
proc finish {} {

        global ns simtype
        $ns flush-trace

        puts "filtering..."
        exec tclsh …/ns-allinone-2.27/nam-1.10/bin/namfilter.tcl ${simtype}.nam
        exec nam ${simtype} &
        exit 0
}

$ns at 0.1 "$ftp start"
$ns at 20.0 "$ftp stop"
$ns at 20.1 "finish"

$ns at 0.0 "$ns trace-annotate \"TCP with multiplicative decrease (Tahoe)\""

$ns run
```

Define finish() procedure

Define init() events

Nam Comment

RUN

II Run TCP Tahoe

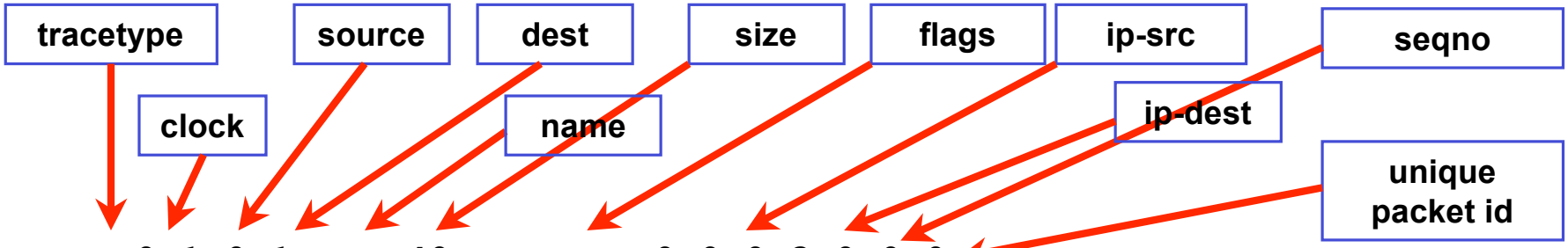
```
$ ls -lh
-rw-r--r--    1 fuessler pi4          1.8K May  3 15:50 tcp-tahoe.tcl

$ time ns tcp-tahoe.tcl
real     0m14.979s
user     0m13.100s
sys      0m0.180s

$ ls -lh
-rw-r--r--    1 fuessler pi4          1.3M May  4 10:15 tcp-tahoe.nam
-rw-r--r--    1 fuessler pi4          1.8K May  3 15:50 tcp-tahoe.tcl
-rw-r--r--    1 fuessler pi4          597K May  4 10:15 tcp-tahoe.tr
```

time shows how long it takes (Xeon 2GHz)

tcp-tahoe.nam → nam Tracefile

tcp-tahoe.tr → packet Tracefile

II The packet tracefile (wired)




```
+ 0.1 0 1 tcp 40 ------- 0 0.0 3.0 0 0
- 0.1 0 1 tcp 40 ------- 0 0.0 3.0 0 0
r 0.120064 0 1 tcp 40 ------- 0 0.0 3.0 0 0
+ 0.120064 1 2 tcp 40 ------- 0 0.0 3.0 0 0
- 0.120064 1 2 tcp 40 ------- 0 0.0 3.0 0 0
r 0.220704 1 2 tcp 40 ------- 0 0.0 3.0 0 0
+ 0.220704 2 3 tcp 40 ------- 0 0.0 3.0 0 0
- 0.220704 2 3 tcp 40 ------- 0 0.0 3.0 0 0
r 0.240768 2 3 tcp 40 ------- 0 0.0 3.0 0 0
...
d 1.371968 1 2 tcp 1040 ------- 0 0.0 3.0 23 38
```

II NAM output

II The NAM tracefile (wired)

```
$tcp tracevar cwnd_
$tcp tracevar ssthresh_
```

Remember the tcl script

```
+ -t 1.747456 -s 1 -d 0 -p ack -e 40 -c 0 -i 63 -a 0 -S 0 -y {22 22}
- -t 1.747456 -s 1 -d 0 -p ack -e 40 -c 0 -i 63 -a 0 -S 0 -f 0 -m 1 -y {22 22}
h -t 1.747456 -s 1 -d 0 -p ack -e 40 -c 0 -i 63 -a 0 -S 0 -y {-1 -1}
r -t 1.75088 -s 1 -d 0 -p ack -e 40 -c 0 -i 60 -a 0 -S 0 -y {22 22}
f -t 1.75088 -s 0 -d 3 -n ssthresh_ -a tcp -v 12 -T v
f -t 1.75088 -s 0 -d 3 -n cwnd_ -a tcp -v 1 -o 24 -T v
```

- **-t: time**
- **-s / -d source and destination**
- **-n parametername**
- **-a agentname**
- **-v / -o value / previous value**
- **-T type (v is normal variable)**

III Extracting information: excursion to regular expression

- specify a pattern that matches text
- like '`rm *.pdf`', but far more powerful
- most UN*X tools are regexp-capable
  - Editors: emacs / vi / kate / nedit…
  - stream editor sed / awk
  - command line tools like grep
  - scripting languages like perl / python / ruby
- Warning: Highly Addictive!

III Extracting information: excursion regular expression 2

History: Stephen Cole Kleene (1909-1994), theory of ‚regular sets' and regular expressions; theory of computation.

.	Dot matches ‚any character'
[ ]	Character class (any out of specified group like [0-9],[+-],[A-Z])
[^]	Negated character class (any but the specified members)
( )	Character group
|	Or bar: matches either expression it separates
*	Quantifier: zero or more
+	Quantifier: one or more
?	Quantifier: zero or one
^	Beginning of line
$	End of line

III Extracting information: excursion regular expression 3

Some Examples:

.*	Matches ‚any character' any times
[0-9]+\.[0-9]+	Matches one or more digits, followed by a dot and again one or more digits
^[0-9]	Matches a digit at the beginning of the line
(cat|dog)	Matches ‚cat' or ‚dog'
^d	Matches a packet drop in a wired trace

III Extracting linformation: simple and fast

```
$ grep ^d tcp-tahoe.tr
d 1.371968 1 2 tcp 1040 ------- 0 0.0 3.0 23 38
d 1.388608 1 2 tcp 1040 ------- 0 0.0 3.0 25 40
d 1.405248 1 2 tcp 1040 ------- 0 0.0 3.0 27 42
d 1.421888 1 2 tcp 1040 ------- 0 0.0 3.0 29 44
d 1.672704 1 2 tcp 1040 ------- 0 0.0 3.0 39 66
d 1.689344 1 2 tcp 1040 ------- 0 0.0 3.0 41 68
d 1.705984 1 2 tcp 1040 ------- 0 0.0 3.0 43 70
d 1.722624 1 2 tcp 1040 ------- 0 0.0 3.0 45 72
d 1.739264 1 2 tcp 1040 ------- 0 0.0 3.0 47 74
d 2.764608 1 2 tcp 1040 ------- 0 0.0 3.0 43 110
d 2.774592 1 2 tcp 1040 ------- 0 0.0 3.0 44 111
d 11.557248 1 2 tcp 1040 ------- 0 0.0 3.0 334 685
d 17.31264 1 2 tcp 1040 ------- 0 0.0 3.0 594 1206
$ grep ^d tcp-tahoe.tr | wc -l
13
```

Select all lines starting with ‚d' (Drops)

wc (counts words)

wc –l (counts lines)

13 dropped packets in scenario

III Extracting information: the power of perl

- perl, i.e. “Practical Extraction and Reporting Language”
- complete scripting language with huge library (http://www.cpan.org)
- powerful string manipulation facilities
- “there are always multiple ways to do things in perl”

  readability / maintainability vs. programming speed

III Perl basics

- scripting language
- available types:
  - scalars (strings), e.g. `$name`
  - arrays (vectors), e.g. `@list` resp. `$list[5]`
  - hashes (associative arrays), e.g. `%dict` resp. `$dict{'name'}`
- syntax similar to C
  - {} delemits blocks
  - semicolon finishes statement
  - if / for / ?: / while / do like in C
- lots of implicit behaviour / special variables etc.

III Perl regular expressions: what's special?

```
#!/usr/bin/perl
$string = "d 1.371968 1 2 tcp 1040 ------- 0 0.0 3.0 23 38";
if ($string =~ /^d (\d+\.\d)\s/){
        print "Drop at time '$1'\n";
};
```

- =~ 'binding' operator followed by regexp in / /
- whole expression is true when matching, false else
- perl know about special abbreviations, e.g.
  - \d : a digit [0-9] \D : not a digit [^0-9]
  - \s : whitespace \S : no whitespace
- Parentheses () also fills special variables $1 - $n
- BTW: grep can be put in perl-mode with grep -P

IV Now let's extract cwnd_ from the nam trace

```
open(NT, "<$namtr") or die "Can't open '$namtr': $!";
...
while (my $line = <NT>){
 chomp($line); # remove newline
 next if not ($line =~ /^f/); # skip lines not starting with f
```

open namtrace

loop over each line

```
f -t 1.75088 -s 0 -d 3 -n cwnd_ -a tcp -v 1 -o 24 -T v
```

```
 if ($line =~ /^f -t (\S+) -s (\S+) -d (\S+) -n cwnd_ -a tcp -v (\S+).*-T (\S+)/){
  my $time = $1;
  my $src_id = $2;
  my $dst_id = $3;
  my $value = $4;
  my $type = $6;
  printf "%f %f\n", $time, $value;
 }else{
  print "# Don't understand line '$line'\n";
 }
}
close(NT);
```

give names to values

print time -> value

IV Extracting information: plot example

```
$ ./getCWND.pl > cwnd.txt
```

generate Data File

```
$ vi plot.gnuplot
```

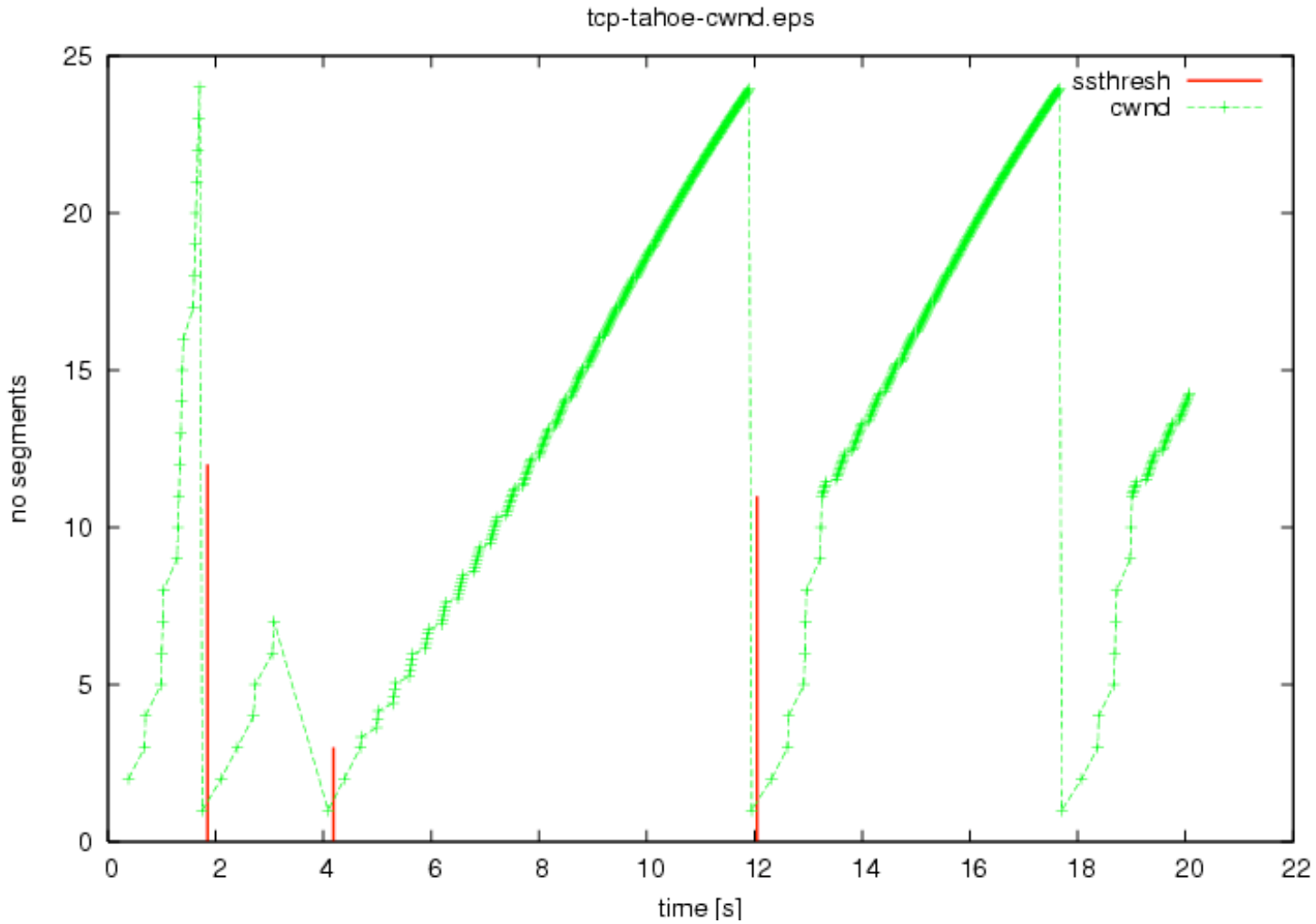
edit gnuplot script

```
$ gnuplot plot.gnuplot
```

run gnuplot script

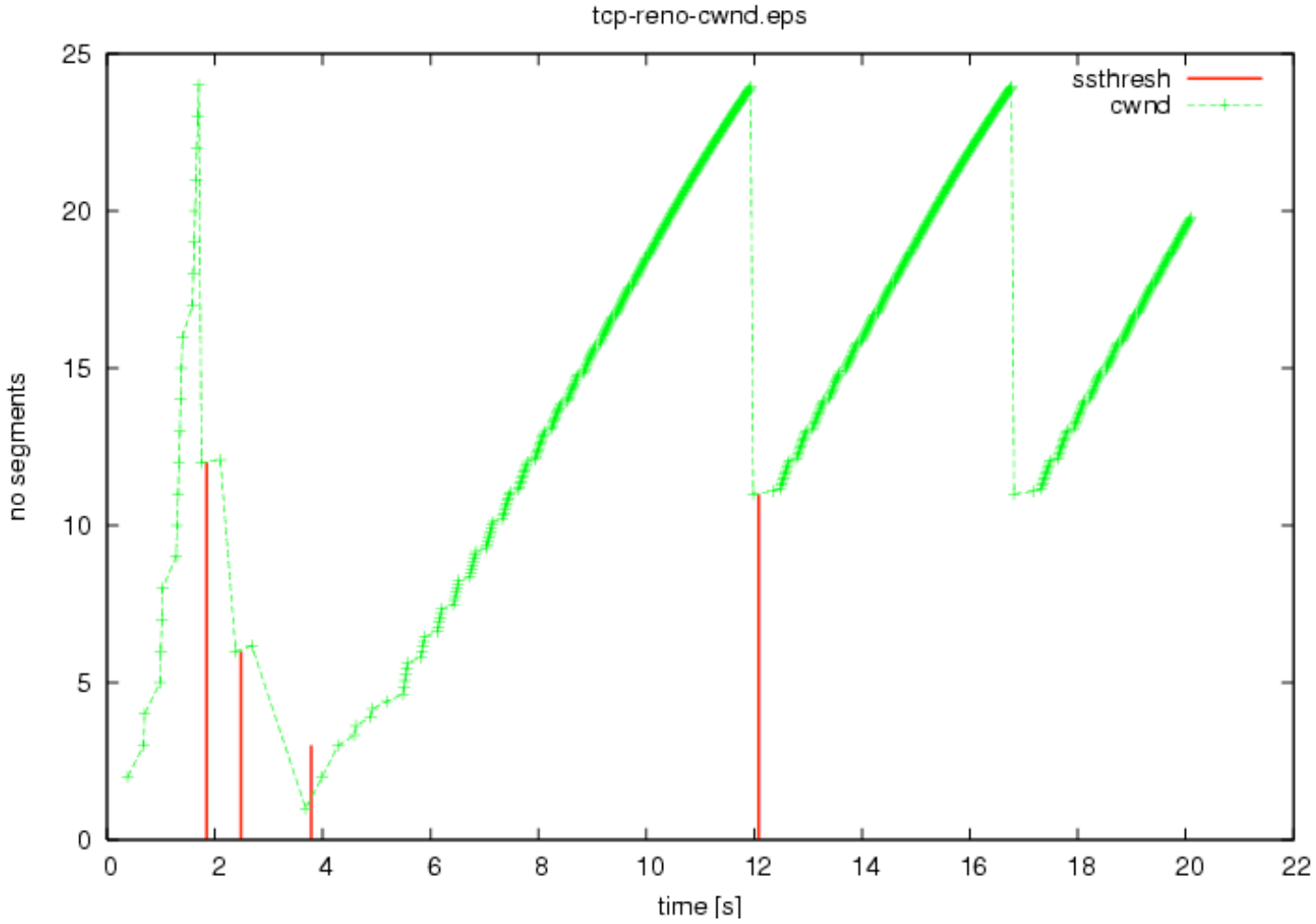
```
set term post eps mono
set out "cwnd.eps"
set title "TCP Tahoe"
set xlabel "time"
set ylabel "cwnd"

plot "cwnd.txt" title "Contention Window Size" with lines
```

IV Resulting graphs: Tahoe



Note: shown is the cwmd whenever it is changed, NOT every packet.

IV Resulting graphs: Reno



IV Observations …

- Tahoe always drops to cwnd of 1
- Reno only drops to ssthresh
- Thus: Overall the bottleneck link seems to be better „exploited“

- … check bandwidth etc. at today‘s NetSim lab.

IV Citation

“I don't know if you can get the flavor of what happens from this description -- it's hard to see without a picture. But I was delighted by how beautifully it worked -- it was like watching the innards of an engine when all the separate motions of crank, pistons and valves suddenly fit together and everything appears in exactly the right place at just the right time.”

Van Jacobson, [Jacobson1990]

IV Summary (educational goals)

- ns-2
  - setup a simple wired scenario
  - create TCP / Application Agents
  - run simulation
- “post-sim”
  - understand tracefiles
  - use grep and perl to extract information
  - make graphs with gnuplot

Literature

- W. Richard Stevens: "*TCP/IP Illustrated – Volume 1*" Addison-Wesley, 1994
- Van Jacobson: "Congestion Avoidance and Control" ACM SIGCOMM Computer and Communication Review, Vol. 18, No. 4, pp. 314-329, 1988
- Van Jacobson: "Modified TCP Congestion Control Algorithm", 1990
- The ns-2 user manual http://www.isi.edu/nsnam/ns
- Jeffrey E.F. Friedl: "Mastering Regular Expressions" O'Reilly 1997
- Randal L. Schwartz, Tom Christiansen, "Learning Perl", 1997