Sesssion3: NS-2 Tracing Files

University of Alexandria
Faculty of Engineering
Computer and Systems Engineering Department

Fourth Year

Networks and Communication Course
NS-2 Network Simulator

Outline

- NS-2 Tracing Files
- Unix command grep and using it with NS-2
- AWK computer language and using it with NS2

Previous Session Revisited

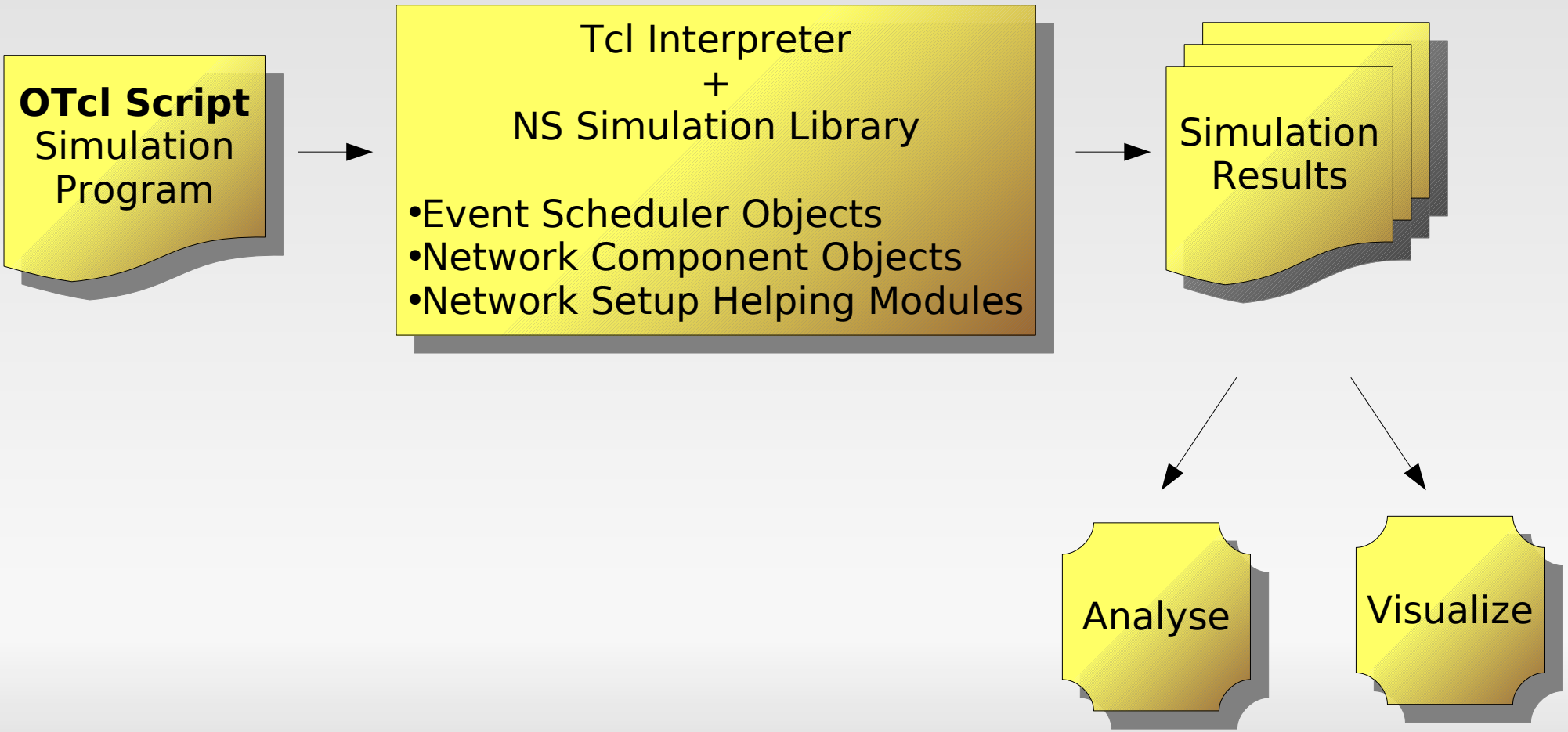
```
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
```

It is a flag to whether or not to introduce random noise.

Previous Session Revisited

Simulation Process at a Glance



Tracing The Network

Preparing the trace file

```
#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf
```

Closing the trace file

```
#Define a 'finish' procedure
proc finish {} {
        global ns tf
        $ns flush-trace
        #Close the Trace file
        close $tf
        exit 0
}
```

NOTE: We can trace subset of event using

```
$ns trace-queue $n2 $n3 $file1
```

Tracing The Network (Cont.)

Tracing File Contents

Each lines consists of:

- Event Descriptor (+, -, d, r)
- Simulation time (in seconds) of that event
- *From* Node & *To* Node, which identify the link on which the event occurred
- Packet type
- Packet size
- Flags (appeared as "------" since no flag is set). Currently, NS implements only the Explicit Congestion Notification (ECN) bit, and the remaining bits are not used.
- Flow id (fid)
- Source and destination address in forms of "node.port".
- The network layer protocol's packet sequence number. *What about UDP?*
- The last field shows the unique id of the packet.

Tracing The Network (Cont.)

Tracing File Format

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)
- : dequeue (at queue)
d : drop    (at queue)
```

```
src_addr : node.port (3.0)
dst_addr : node.port (0.0)
```

```
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
```

We need to extract the necessary data out of this file. How?

grep

- It is a **UNIX** command to filter a file.

- The name comes from “search **g**lobally for lines matching the **r**egular **e**xpression and **p**rint them”

- **grep** takes a regular expression on the command line, reads the standard input or list of files, and outputs the lines containing matches for the regular expression

grep (Cont.)

Example

```
grep “@” file1.txt > file2.txt
```

Using grep with NS-2

Example 1

In the trace file, if you are intersted only in data concerning `tcp` packtes that went from node 0 to node 2

```
grep “0 2 tcp” tr1.tr > tr2.tr
```

Using grep with NS-2 (Cont.)

Example 2

In the trace file, if you are intersted only in the received packtes

```
grep "^r" tr1.tr > tr2.tr
```

Using grep with NS-2 (Cont.)

Example 3

In the trace file, if you are intersted only in lines the begin with “s” and have “tcp 1020” later

```
grep “^r” tr1.tr | grep “tcp 1020” > tr2.tr
```

AWK

- **AWK** is a general purpose *computer language*

- **AWK** is designed for processing text-based data, either in files or data streams

- The name **AWK** is derived from the surnames of its authors — Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan;

Computer languages include:

* Programming languages (e.g. C++)
* Scripting languages
* Specification languages
* Query language (e.g. SQL, XQuery)
* Markup languages (e.g. HTML - typically used for producing documents)
* Transformation languages (e.g., XSLT)
* Template processing languages
* 4GL (4^{th} Generation Language)
* Hardware description languages

AWK (Cont.)

How to run AWK file?

```
awk -f file1.awk file2.txt

awk -f file1.awk file2.txt > out.txt
```

`file1.awk` : is a command file
`file2.txt` : is a primary input file
`out.txt` : is an output file

AWK (Cont.)

A typical AWK program consists of a series of lines, each of them is on the form

`/pattern/ { action }`

Pattern is a regular expression
Action is a command.

- Most implementations of AWK use extended regular expressions by default.
- AWK looks through the input file; when it finds a line that matches pattern, it executes the command(s) specified in action.

Other line forms:

`BEGIN { action }`

Executes action commands at the beginning of the script execution,

`END { action }`

Executes action commands after the end of input.

`/pattern/`

Prints any lines matching pattern.

`{ action }`

Executes action for each line in the input.

AWK (Cont.)

Example 1

```
{ print $1 }
```

AWK (Cont.)

Example 2

```
BEGIN { FS = "\t" }
{ nl++ }
{ s=s+$4 }
END { print "average = " s/nl }
```

FS : is the Field Separator.
OFS : is the Output Field Separator

AWK (Cont.)

Example 3

```
{ w += NF; c += length}
END { print NR, w, c }
```

It prints the number of lines, number of words and number of characters in the file

AWK (Cont.)

Example 4
AWK supports associative arrays

```
BEGIN { FS="[^a-zA-Z]+"}

{ for (i=1; i<=NF; i++)
      words[tolower($i)]++
}

END { for (i in words)
     print i, words[i]
}
```

It gets the frequencies of the words.

AWK (Cont.)

Example 5
Functions in AWK

```
function Square (number, temp) {
    temp = number * number
    return temp
}
.
.
.
.
print Square(9)      # Prints 81
```

Note:
Functions can have variables that are in the local scope. The names of these are added to the end of the argument list, though values for these should be omitted when calling the function.

Using AWK with NS-2

```
#This program is used to calculate the packet loss rate between nodes
# 1,2 for the application having the flow id = 2
BEGIN {
# Initialization. fsDrops: packets drop. numFs: packets sent
        fsDrops = 0;
        numFs = 0;
}
{
   action = $1;      time = $2;
   from = $3;        to = $4;
   type = $5;        pktsize = $6;
   flow_id = $8;     src = $9;
   dst = $10;        seq_no = $11;
   packet_id = $12;
        if (from==1 && to==2 && action == "+")
                numFs++;
        if (flow_id==2 && action == "d")
                fsDrops++;
}
END {
        printf("number of packets sent:%d lost:%d\n", numFs, fsDrops);
}
```

I'm ready for questions, are you?