Using Neural Networks and Statistical Machinery to improve remote OS Detection

Javier Burroni - Carlos Sarraute

Core Security Technologies

HITBSecConf 2006

OUTLINE

1. Introduction

2. DCE-RPC Endpoint mapper

3. OS Detection based on Nmap signatures

4. Dimension reduction and training

5. Experiment with Cron-OS

1. Introduction

2. DCE-RPC Endpoint mapper

3. OS Detection based on Nmap signatures

4. Dimension reduction and training

5. Experiment with Cron-OS

OS Identification

- OS Identification = OS Detection = OS Fingerprinting
- Crucial step of the penetration testing process
  - actively send test packets and study host response
- First generation: analysis of differences between TCP/IP stack implementations
- Next generation: analysis of application layer data (DCE RPC endpoints)
  - to refine detection of Windows versions / editions / service packs

Limitations of OS Fingerprinting tools

- Some variation of “best fit” algorithm is used to analyze the information
  - will not work in non standard situations
  - inability to extract key elements
- Our proposal:
  - focus on the technique used to analyze the data
  - we have developed tools using neural networks
  - successfully integrated into commercial software

1. Introduction

2. DCE-RPC Endpoint mapper

3. OS Detection based on Nmap signatures

4. Dimension reduction and training

5. Experiment with Cron-OS

Windows DCE-RPC service

- By sending an RPC query to a host’s port 135
  you can determine which services or programs are registered

- Response includes:
  - UUID = universal unique identifier for each program
  - Annotated name
  - Protocol that each program uses
  - Network address that the program is bound to
  - Program’s endpoint

Endpoints for a Windows 2000 Professional edition service pack 0

- uuid="5A7B91F8-FF00-11D0-A9B2-00C04FB6E6FC" annotation="Messenger Service"
  - protocol="ncalrpc" endpoint="ntsvcs" id="msgsvc.1"
  - protocol="ncacn_np" endpoint="\PIPE\ntsvcs" id="msgsvc.2"
  - protocol="ncacn_np" endpoint="\PIPE\scerpc" id="msgsvc.3"
  - protocol="ncadg_ip_udp" id="msgsvc.4"

- uuid="1FF70682-0A51-30E8-076D-740BE8CEE98B"
  - protocol="ncalrpc" endpoint="LRPC" id="mstask.1"
  - protocol="ncacn_ip_tcp" id="mstask.2"

- uuid="378E52B0-C0A9-11CF-822D-00AA0051E40F"
  - protocol="ncalrpc" endpoint="LRPC" id="mstask.3"
  - protocol="ncacn_ip_tcp" id="mstask.4"

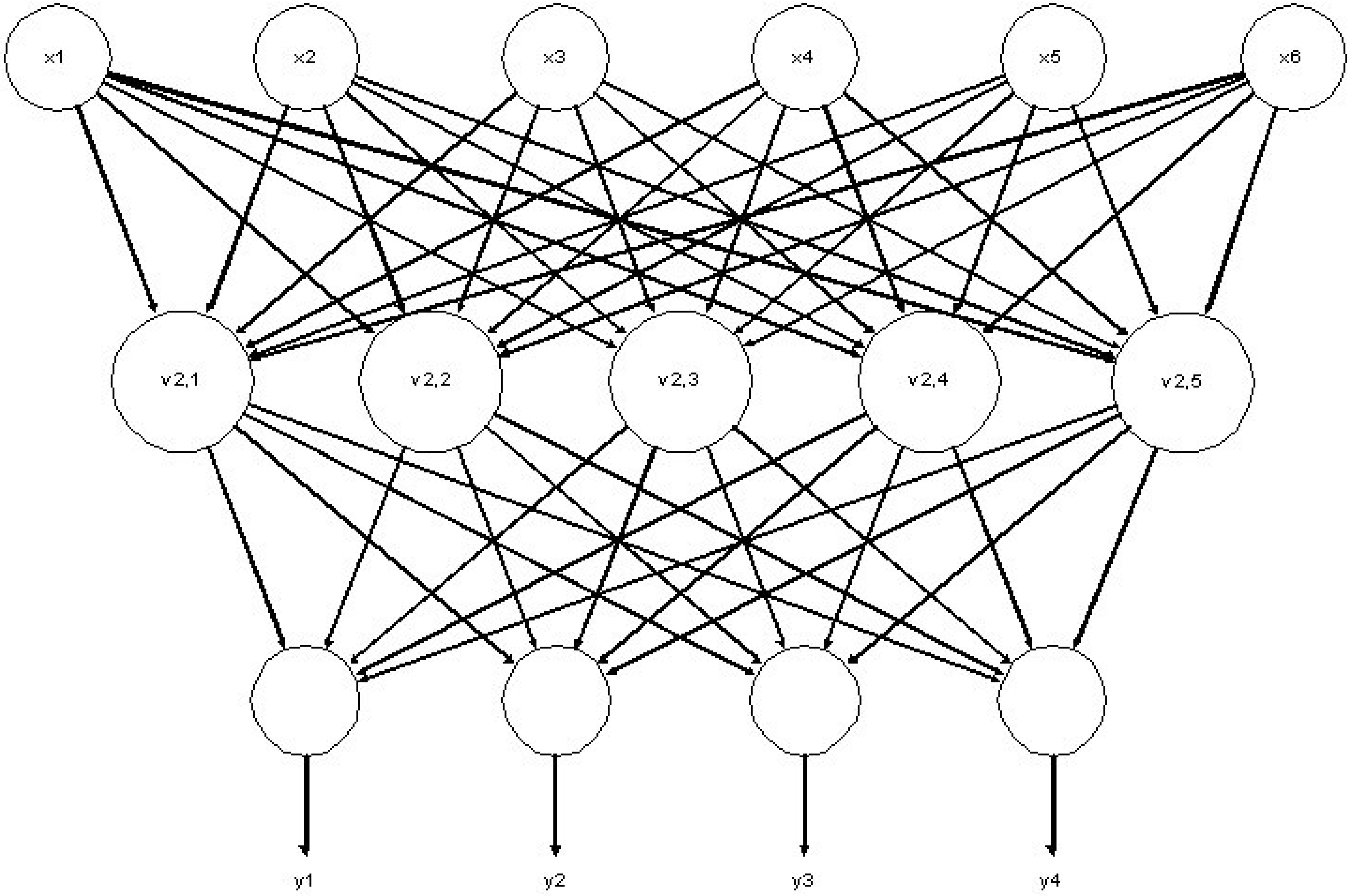
Neural networks come into play…

- It’s possible to distinguish Windows versions, editions and service packs based on the combination of endpoints provided by DCE-RPC service

- Idea: model the function which maps endpoints combinations to OS versions with a multilayer perceptron neural network

- Several questions arise:
  - what kind of neural network do we use?
  - how are the neurons organized?
  - how do we map endpoints combinations to neural network inputs?
  - how do we train the network?

Multilayer Perceptron Neural Network



413 neurons

42 neurons

25 neurons

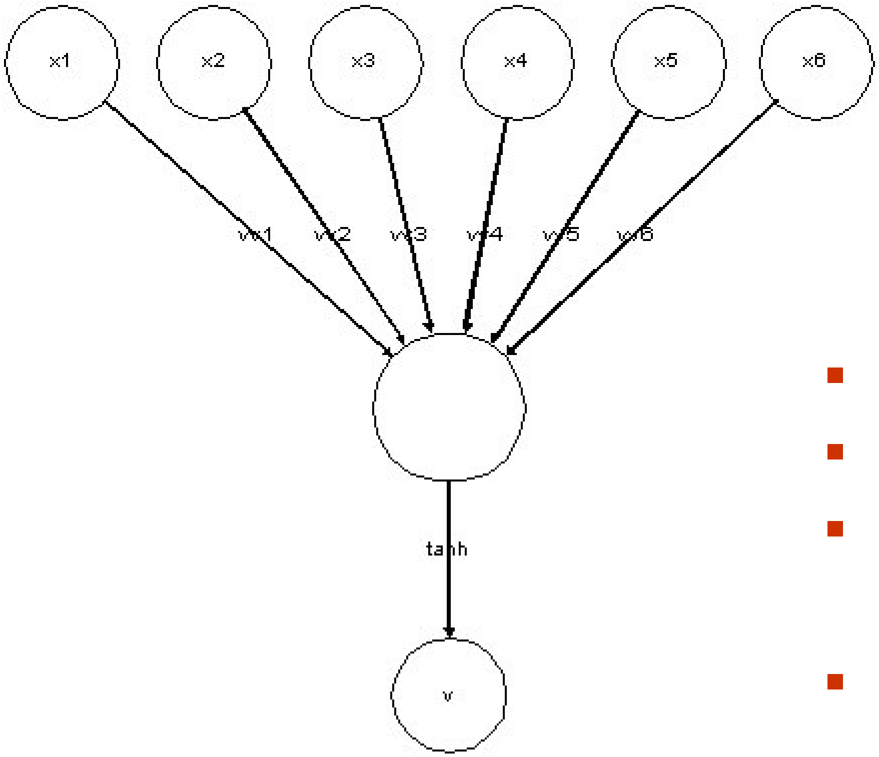
3 layers topology

- Input layer : 413 neurons
  - one neuron for each UUID
  - one neuron for each endpoint corresponding to the UUID
  - handle with flexibility the appearance of an unknown endpoint

- Hidden neuron layer : 42 neurons
  - each neuron represents combinations of inputs

- Output layer : 25 neurons
  - one neuron for each Windows version and edition
    - » Windows 2000 professional edition
  - one neuron for each Windows version and service pack
    - » Windows 2000 service pack 2
  - errors in one dimension do not affect the other

What is a perceptron?



$$v_{i,j} = f\left(\sum_{k=0}^{n} w_{i,j,k} \cdot x_k \right)$$

- $x_1 \ldots x_n$ are the inputs of the neuron
- $w_{i,j,0} \ldots w_{i,j,n}$ are the weights
- f is a non linear activation function
  - we use hyperbolic tangent tanh
- $v_{i,j}$ is the output of the neuron

Training of the network = finding the weights for each neuron

Back propagation algorithm

- Training using the back-propagation algorithm:
- for the output layer
  - given an expected output $y_1 \dots y_m$
  - calculate an estimation of the error

$$\delta_{i,j} = f'(v_{i,j})\,(y_j - v_{i,j})$$

- this is propagated to the previous layers as:

$$\delta_{i,j} = f'(v_{i,j}) \sum_k w_{i,j,k} \cdot \delta_{i+1,j}$$

Computing new weights

- The new weights, at time *t+1*, are:

$$w_{i,j,k}(t+1) = w_{i,j,k}(t) + \Delta w_{i,j,k}(t)$$

- where:

$$\Delta w_{i,j,k}(t) = (\lambda \cdot \delta_{i+1,k} \cdot v_{i,j}) + \mu \cdot \Delta w_{i,j,k}(t-1)$$

learning rate

momentum

Supervised training

- We have a dataset with inputs and expected outputs
- One generation: recalculate weights for each input / output pair
- Complete training = 10350 generations
- For each generation of the training process, inputs are reordered randomly (so the order does not affect training)

Sample result of the Impact module

```
Neural Network Output (close to 1 is better):
Windows NT4: 4.87480503763e-005
Editions:
    Enterprise Server: 0.00972694324639
    Server: -0.00963500026763
Service Packs:
    6: 0.00559659167371
    6a: -0.00846224120952
Windows 2000: 0.996048928128
Editions:
    Server: 0.977780526016
    Professional: 0.00868998746624
    Advanced Server: -0.00564873813703
Service Packs:
    4: -0.00505441088081
    2: -0.00285674134367
    3: -0.00936655583402
    0: -0.00320117552666
    1: 0.921351036343
```

Sample result of the Impact module (cont.)

```
Windows 2003: 0.00302898647853
Editions:
    Web Edition: 0.00128127138728
    Enterprise Edition: 0.00771786077082
    Standard Edition: -0.0077145024893
Service Packs:
    0: 0.000853988551952
Windows XP: 0.00605168045887
Editions:
    Professional: 0.00115635710749
    Home: 0.000408057333416
Service Packs:
    2: -0.00160404945542
    0: 0.00216065240615
    1: 0.000759109188052
Setting OS to Windows 2000 Server sp1
Setting architecture: i386
```

Result comparison

- Results of our laboratory:

	Old DCE-RPC module	DCE-RPC with neural networks
Perfect matches	6	7
Partial matches	8	14
Mismatches	7	0
No match	2	2

1. Introduction

2. DCE-RPC Endpoint mapper

3. OS Detection based on Nmap signatures

4. Dimension reduction and training

5. Experiment with Cron-OS

Nmap tests

- Nmap is a network exploration tool and security scanner
- includes OS detection based on the response of a host to 9 tests

Test	send packet	to port	with flags enabled
T1	TCP	open TCP	SYN, ECN-Echo
T2	TCP	open TCP	no flags
T3	TCP	open TCP	URG, PSH, SYN, FIN
T4	TCP	open TCP	ACK
T5	TCP	closed TCP	SYN
T6	TCP	closed TCP	ACK
T7	TCP	closed TCP	URG, PSH, FIN
PU	UDP	closed UDP	
TSeq	TCP * 6	open TCP	SYN

Nmap signature database

- Our method is based on the Nmap signature database
- A signature is a set of rules describing how a specific version / edition of an OS responds to the tests. Example:

```
# Linux 2.6.0-test5 x86
Fingerprint Linux 2.6.0-test5 x86
Class Linux | Linux | 2.6.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<2D3CFA0&>73C6B%IPID=Z%TS=1000HZ)
T1(DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=Y%DF=Y%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%UL
  EN=134%DAT=E)
```

Wealth and weakness of Nmap

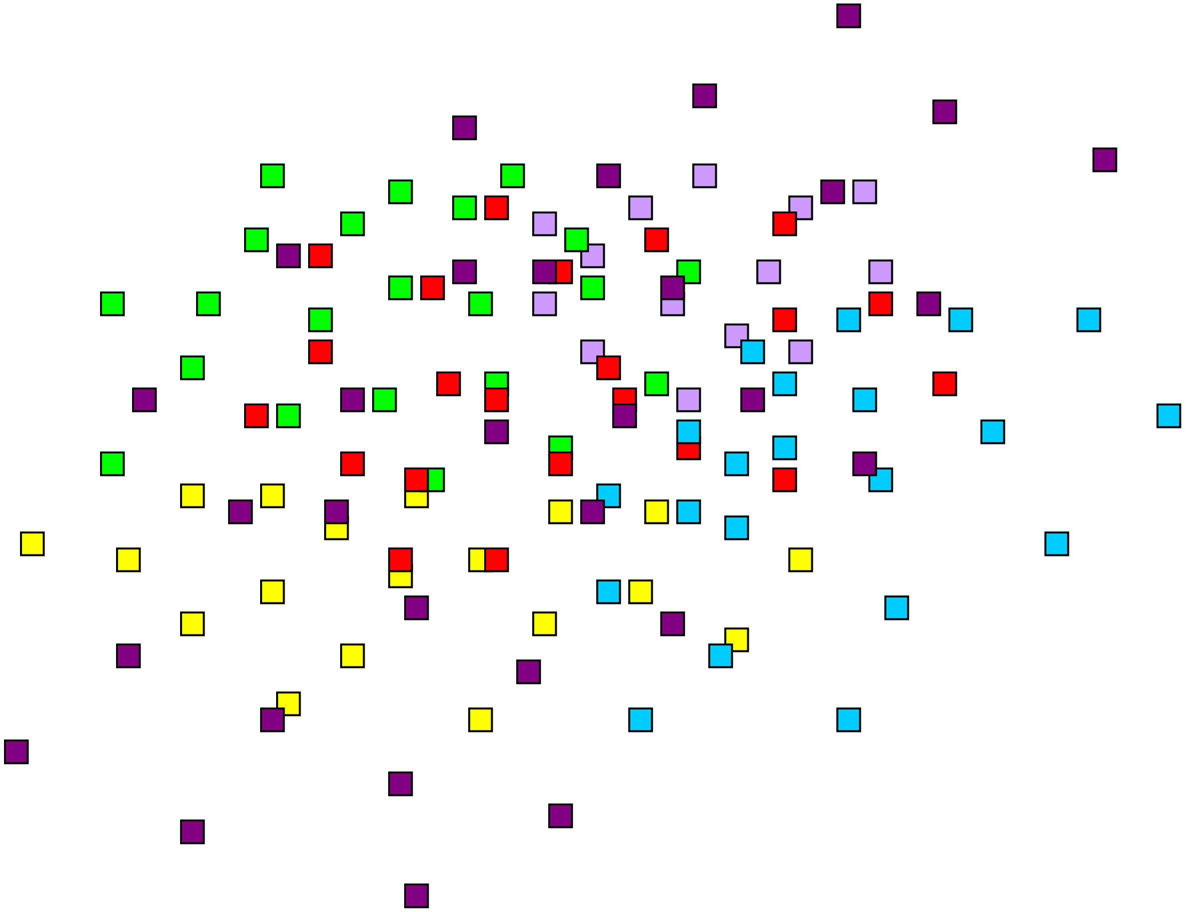
- Nmap database contains 1684 signatures

- Nmap works by comparing a host response to each signature in the database:
  - a score is assigned to each signature
  - score = number of matching rules / number of considered rules
  - “best fit” based on Hamming distance

- Problem: improbable operating systems
  - generate less responses to the tests
  - and get a better score!
  - e.g. a Windows 2000 version detected as Atari 2600 or HPUX …

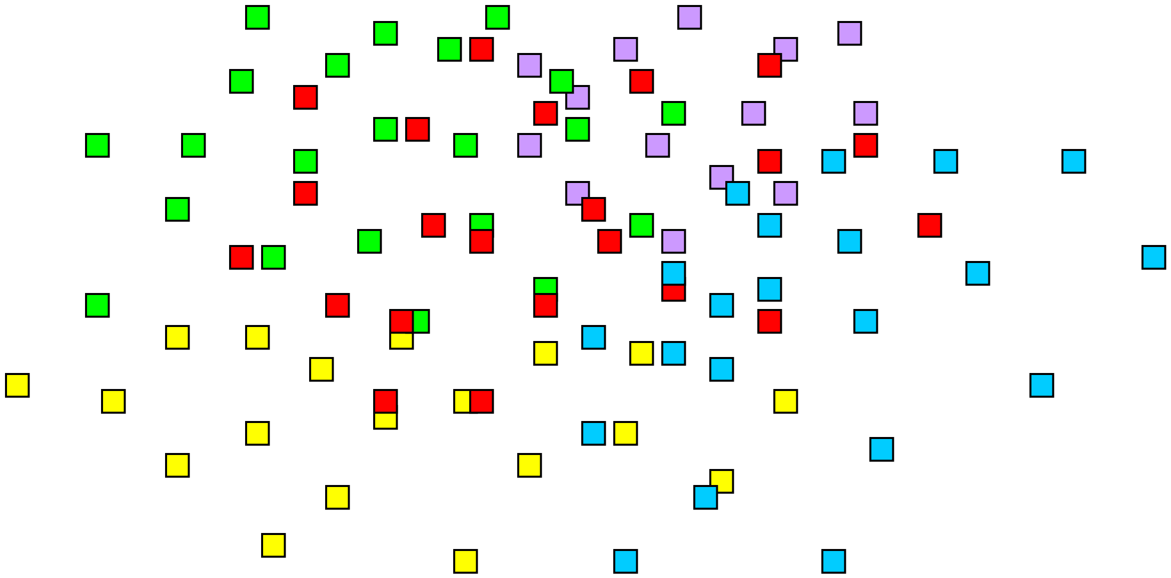
Symbolic representation of the OS space

- The space of host responses has 560 dimensions
- Colors represents different OS families

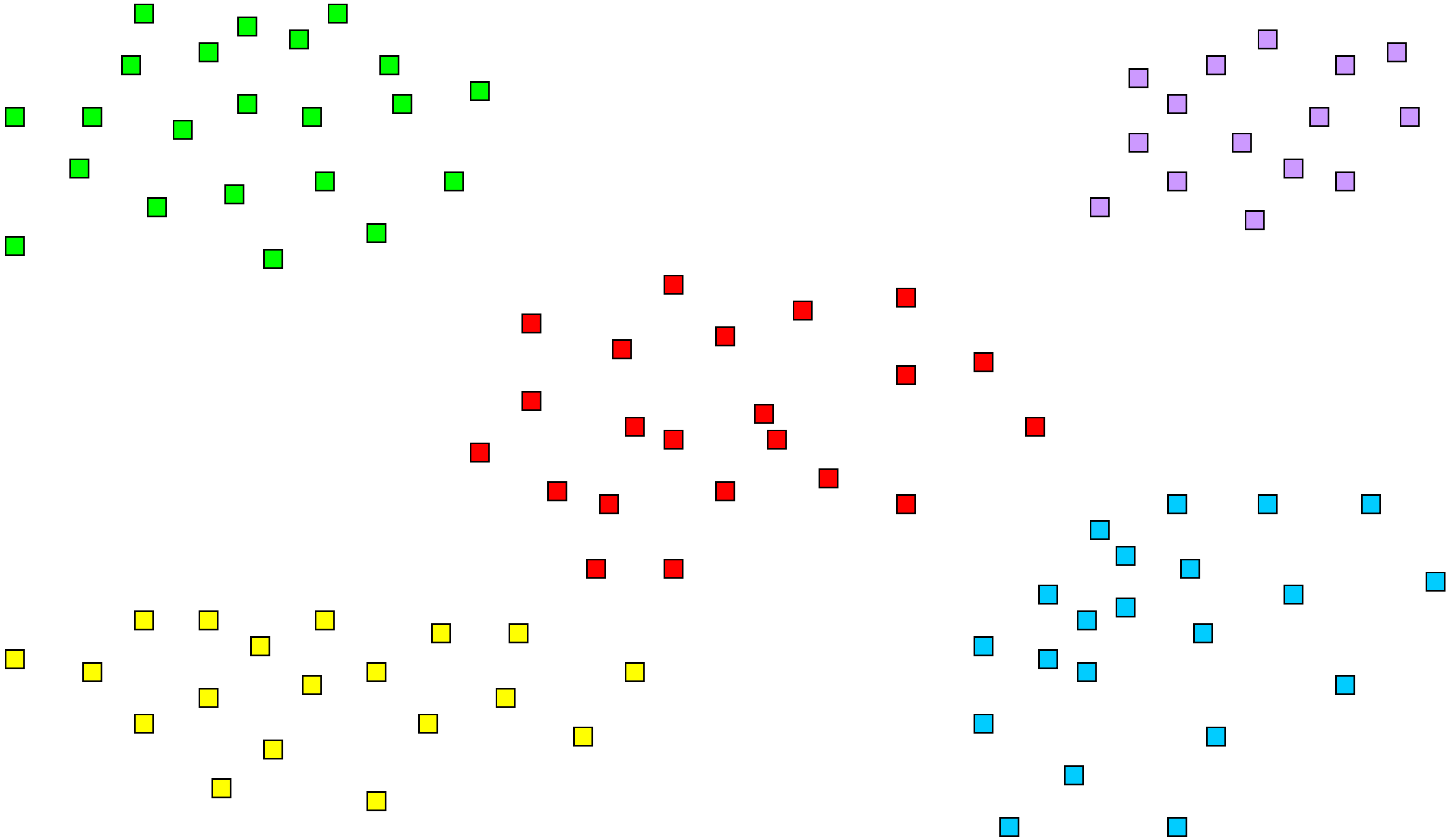


Picture after filtering irrelevant OS

- OS detection is a step of the penetration test process
  - we only want to detect Windows, Linux, Solaris, OpenBSD, FreeBSD, NetBSD

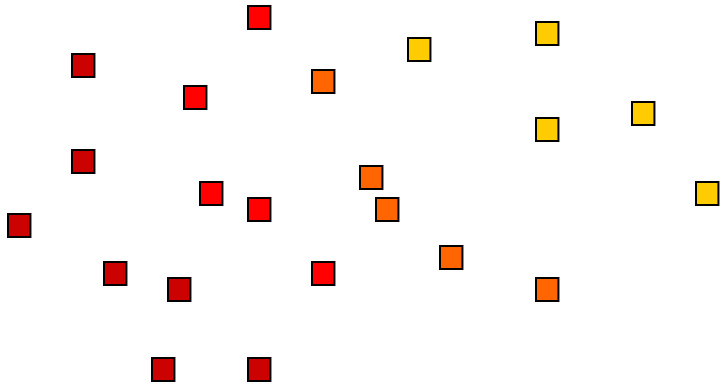


Picture after separating the OS families



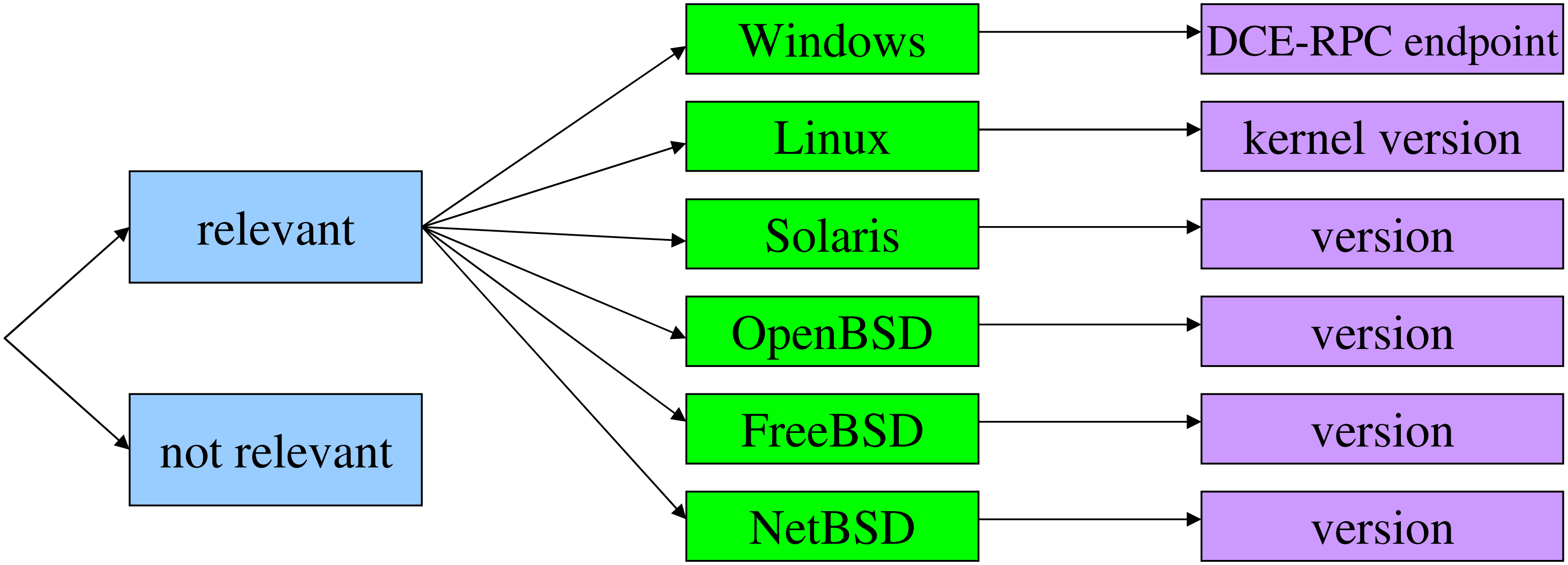
Distinguish versions within each OS family

- The analysis to distinguish different versions is done after we know the family
  - for example, we know that the host is running OpenBSD and want to know the version



Hierarchical Network Structure

- Analyze the responses with different neural networks
- Each analysis is conditionned by the results of the previous analysis



So we have 5 neural networks…

- One neural network to decide if the OS is relevant / not relevant

- One neural network to decide the OS family:
  - Windows, Linux, Solaris, OpenBSD, FreeBSD, NetBSD

- One neural network to decide Linux version
- One neural network to decide Solaris version
- One neural network to decide OpenBSD version

- Each neural network requires special topology design and training!
  - OpenBSD version network is trained with a dataset containing only OpenBSD host responses

Neural Network inputs

- Assign a set of inputs neurons for each test
- Details for tests T1 … T7:

- one neuron for ACK flag
  - one neuron for each response: S, S++, O
- one neuron for DF flag
  - one neuron for response: yes/no
- one neuron for Flags field
  - one neuron for each flag: ECE, URG, ACK, PSH, RST, SYN, FIN
- 10 groups of 6 neurons for Options field
  - we activate one neuron in each group according to the option
    EOL, MAXSEG, NOP, TIMESTAMP, WINDOW, ECHOED
- one neuron for W field (window size)

Example of neural network inputs

- For flags or options: input is 1 or -1 (present or absent)
- Others have numerical input
  - the W field (window size)
  - the GCD (greatest common divisor of initial sequence numbers)

- Example of Linux 2.6.0 response:

```
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
```

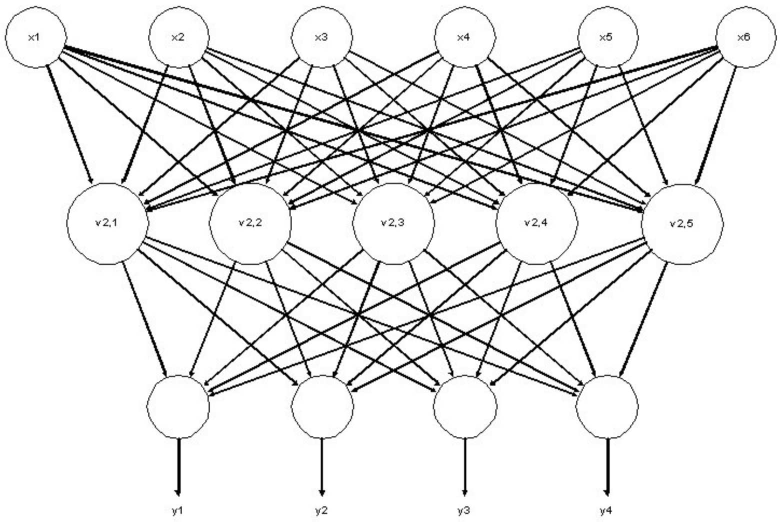
- maps to:

ACK	S	S++	O	DF	Yes	Flags	E	U	A	P	R	S	F	…
1	-1	1	-1	1	1	1	-1	-1	1	-1	-1	1	-1	…

Neural network topology

- Input layer of 560 dimensions
  - lots of redundancy
  - gives flexibility when faced to unknown responses
  - but raises performance issues!
  - dimension reduction is necessary…

- 3 layers neural network, for example the first neural network (relevant / not relevant filter) has:



input layer : 96 neurons

hidden layer : 20 neurons

output layer : 1 neuron

Dataset generation

- To train the neural network we need
  - inputs (host responses)
  - with corresponding outputs (host OS)
- Signature database contains 1684 rules
  - a population of 15000 machines needed to train the network!
  - we don’t have access to such population…
  - scanning the Internet is not an option!

- Generate inputs by Monte Carlo simulation
  - for each rule, generate inputs matching that rule
  - number of inputs depends on empirical distribution of OS
    - » based on statistical surveys
  - when the rule specifies options or range of values
    - » chose a value following uniform distribution

1. Introduction

2. DCE-RPC Endpoint mapper

3. OS Detection based on Nmap signatures

4. Dimension reduction and training

5. Experiment with Cron-OS

Inputs as random variables

- We have been generous with the input
  - 560 dimensions, with redundancy
  - inputs dataset is very big
  - the training convergence is slow…

- Consider each input dimension as a random variable X_i
  - input dimensions have different orders of magnitude
    - flags take 1/-1 values
    - the ISN (initial sequence number) is an integer
  - normalize the random variables:

$$\frac{X_i - \mu_i}{\sigma_i}$$

expected value

standard deviation

Correlation matrix

- We compute the correlation matrix *R*:

$$R_{i,j} = \frac{E[(X_i - \mu_i)(X_j - \mu_j)]}{\sigma_i \, \sigma_j}$$

- After normalization this is simply:

$$R_{i,j} = E[X_i \, X_j]$$

expected value

- The correlation is a dimensionless measure of statistical dependence
  - closer to 1 or -1 indicates higher dependence
  - linear dependent columns of *R* indicate dependent variables
  - we keep one and eliminate the others
  - constants have zero variance and are also eliminated

Example of OpenBSD fingerprints

```
Fingerprint OpenBSD 3.6 (i386)
Class OpenBSD | OpenBSD | 3.X | general purpose
T1(DF=N%W=4000%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=N)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

Fingerprint OpenBSD 2.2 - 2.3
Class OpenBSD | OpenBSD | 2.X | general purpose
T1(DF=N%W=402E%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=N%W=402E%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=4000%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
```

Relevant fields to distinguish OpenBSD versions

New index	Original index	Field name
0	20	T1:TCP_OPT_1_EOL
1	26	T1:TCP_OPT_2_EOL
2	29	T1:TCP_OPT_2_TIMESTAMP
3	74	T1:W_FIELD
4	75	T2:ACK_FIELD
5	149	T2:W_FIELD
6	150	T3:ACK_FIELD
7	170	T3:TCP_OPT_1_EOL
8	179	T3:TCP_OPT_2_TIMESTAMP
9	224	T3:W_FIELD
10	227	T4:SEQ_S
11	299	T4:W_FIELD
12	377	T6:SEQ_S
13	452	T7:SEQ_S

Relevant fields to distinguish OpenBSD versions (cont.)

New index	Original index	Field name
14	525	TSeq:CLASS_FIELD
15	526	TSeq:SEQ_TD
16	528	TSeq:SEQ_RI
17	529	TSeq:SEQ_TR
18	532	TSeq:GCD_FIELD
19	533	TSeq:IPID_FIELD
20	535	TSeq:IPID_SEQ_BROKEN_INCR
21	536	TSeq:IPID_SEQ_RPI
22	537	TSeq:IPID_SEQ_RD
23	540	TSeq:SI_FIELD
24	543	TSeq:TS_SEQ_2HZ
25	546	TSeq:TS_SEQ_UNSUPPORTED
26	555	PU:UCK_RID_RIPCK_EQ
27	558	PU:UCK_RID_RIPCK_ZERO

Principal Component Analysis (PCA)

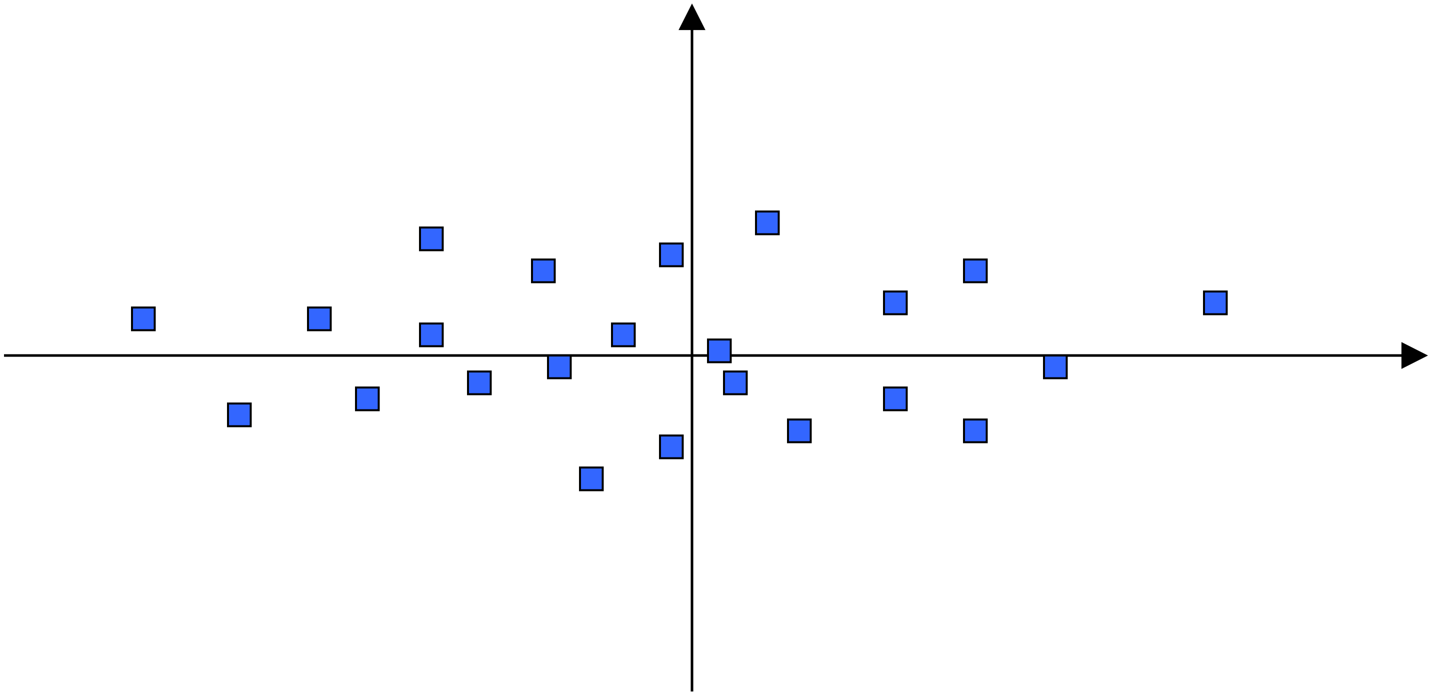
- Further reduction involves Principal Component Analysis (PCA)

- Idea: compute a new basis (coordinates system) of the input space
  - the greatest variance of any projection of the dataset in a subspace of k dimensions
  - comes by projecting to the first k basis vectors

- PCA algorithm:
  - compute eigenvectors and eigenvalues of R
  - sort by decreasing eigenvalue
  - keep first k vectors to project the data
  - parameter k chosen to keep 98% of total variance

Idea of Principal Component Analysis

- Keep the dimensions which have higher variance
  - higher eigenvalues of the Correlation Matrix



Resulting neural network topology

- After performing these reductions we obtain the following neural network topologies (original input size was 560 in all cases)

Analysis	Input layer (after correlation matrix reduction)	Input layer (after PCA)	Hidden layer	Output layer
Relevance	204	96	20	1
Operating System	145	66	20	6
Linux	100	41	18	8
Solaris	55	26	7	5
OpenBSD	34	23	4	3

Adaptive learning rate

- Strategy to speed up training convergence

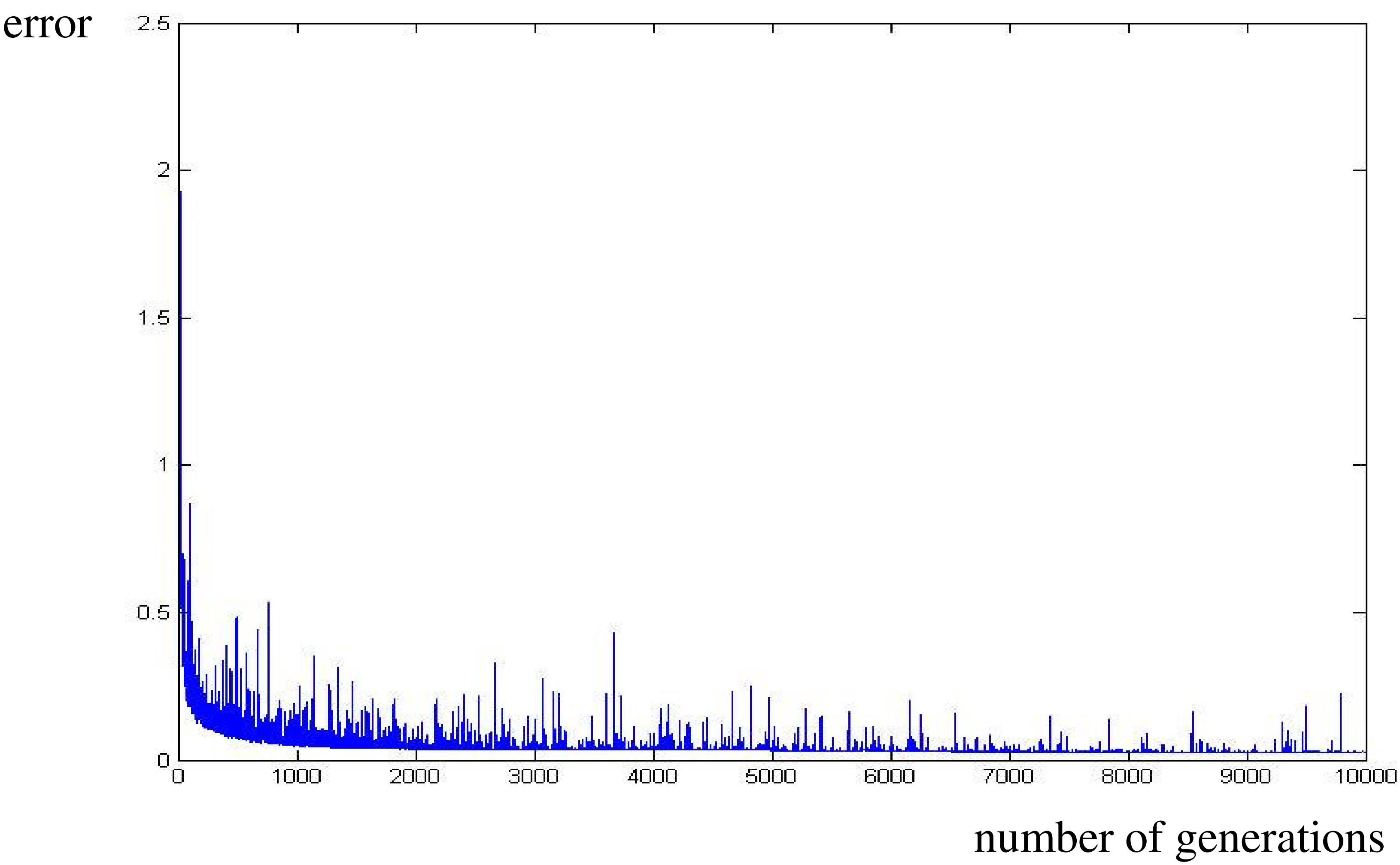
- Calculate the quadratic error estimation
  (y_i are the expected outputs, v_i are the actual outputs):

$$\frac{\sum_{i=1}^{n}(y_i - v_i)^2}{n}$$

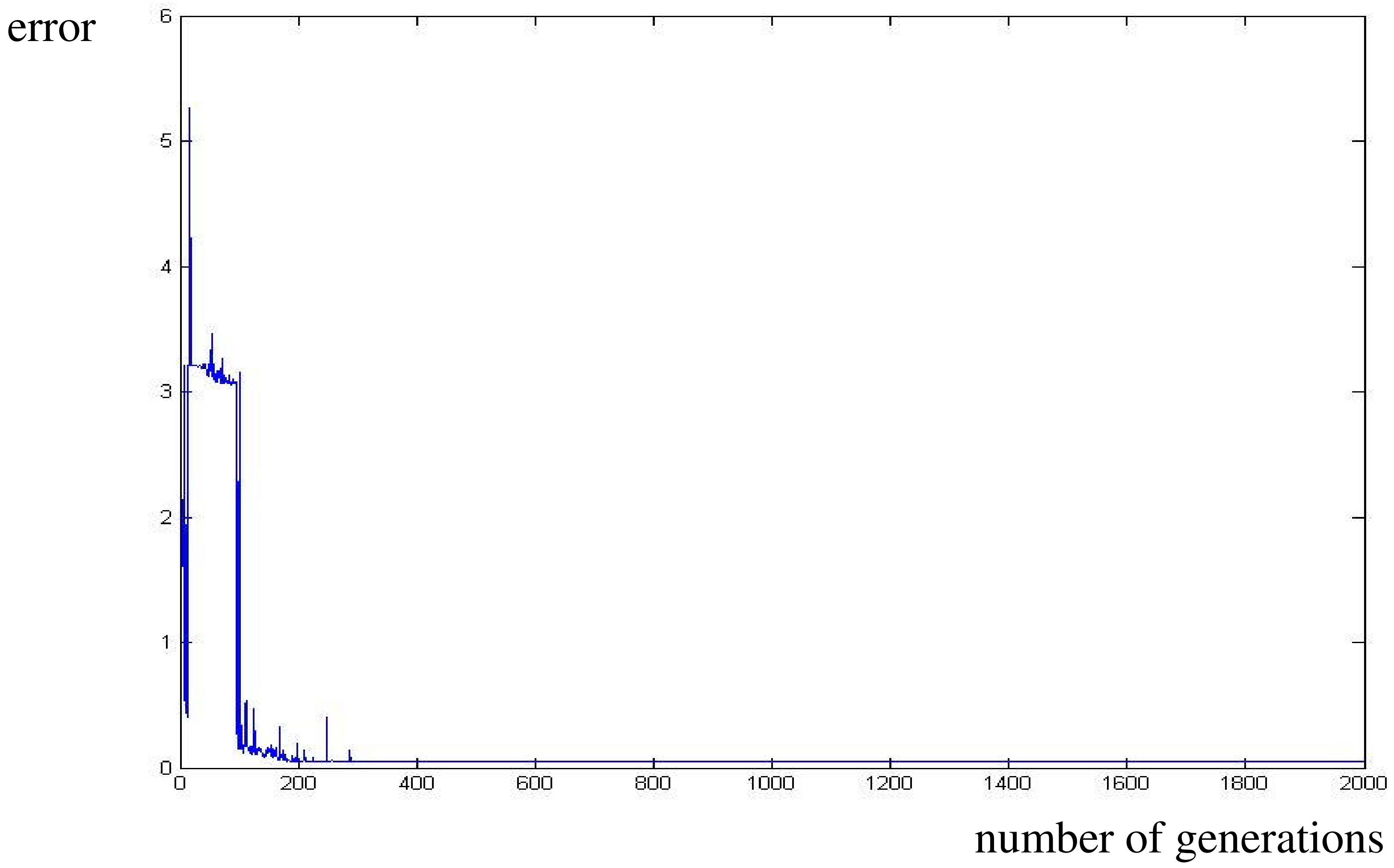
- Between generations (after processing all dataset input/output pairs)
  - if error is smaller then increase learning rate
  - if error is bigger then decrease learning rate

- Idea: move faster if we are in the correct direction

Error evolution (fixed learning rate)



Error evolution (adaptive learning rate)



Sample result (host running Solaris 8)

- Relevant / not relevant analysis

```
    0.99999999999999789          relevant
```

- Operating System analysis

```
    -0.99999999999999434         Linux
    0.99999999921394744          Solaris
    -0.99999999999998057         OpenBSD
    -0.99999964651426454         FreeBSD
    -1.0000000000000000          NetBSD
    -1.0000000000000000          Windows
```

- Solaris version analysis

```
    0.98172780325074482          Solaris 8
    -0.99281382458335776         Solaris 9
    -0.99357586906143880         Solaris 7
    -0.99988378968003799         Solaris 2.X
    -0.99999999977837983         Solaris 2.5.X
```

1. Introduction

2. DCE-RPC Endpoint mapper

3. OS Detection based on Nmap signatures

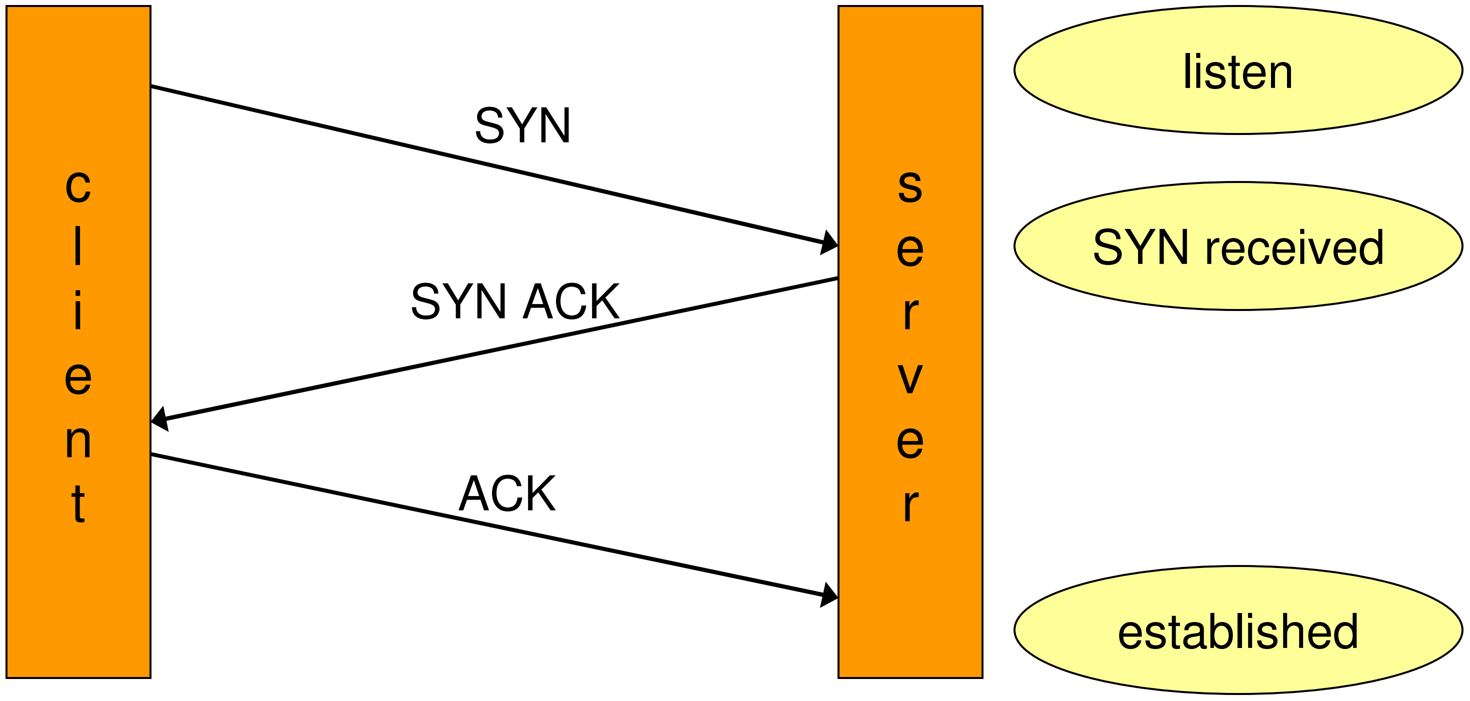
4. Dimension reduction and training

5. Experiment with Cron-OS

Cron-OS idea

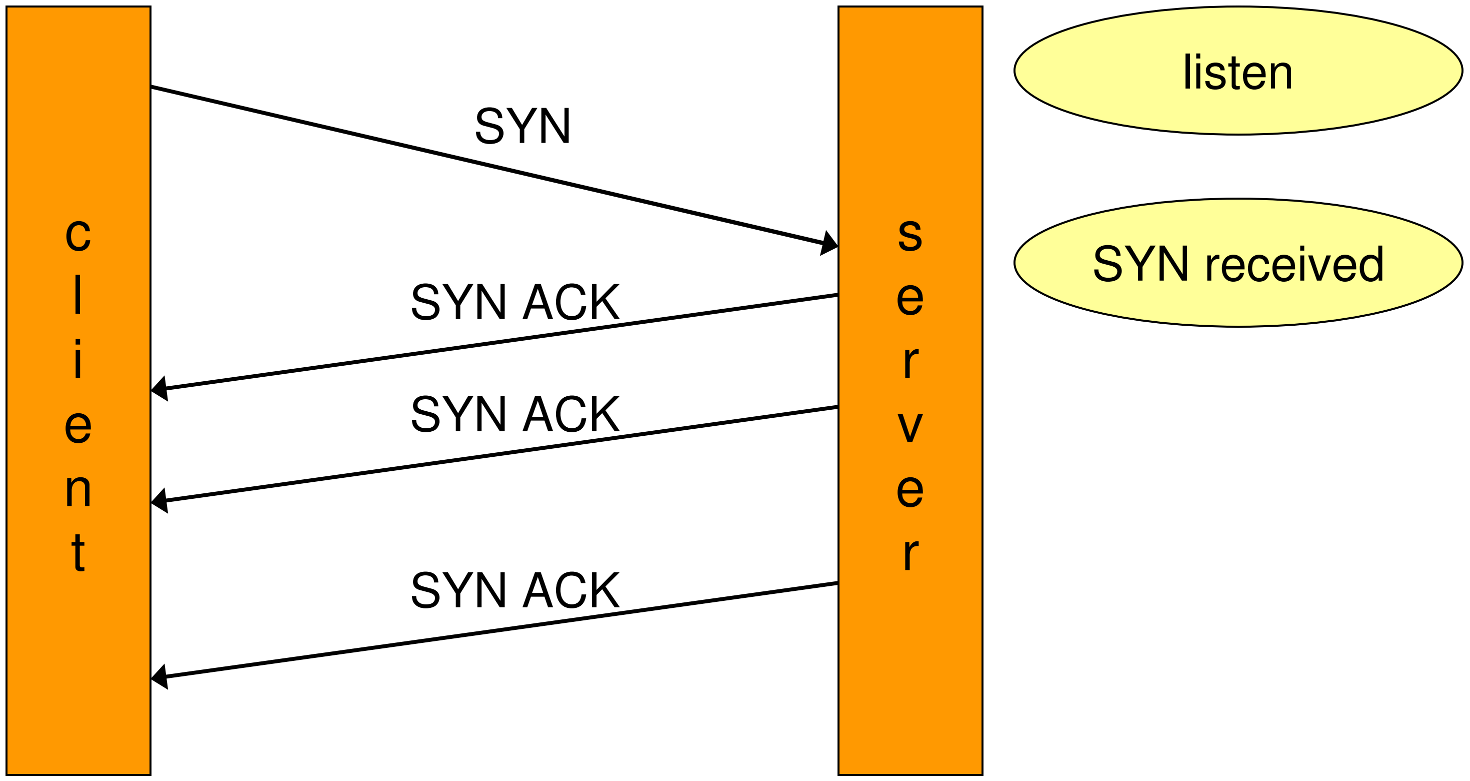
- Tool by Olivier Courtay, Olivier Heen, and Franck Veysset
- Implemented as an extension of Nmap: nmap-Cron-OS

- Normal 3-way handshake



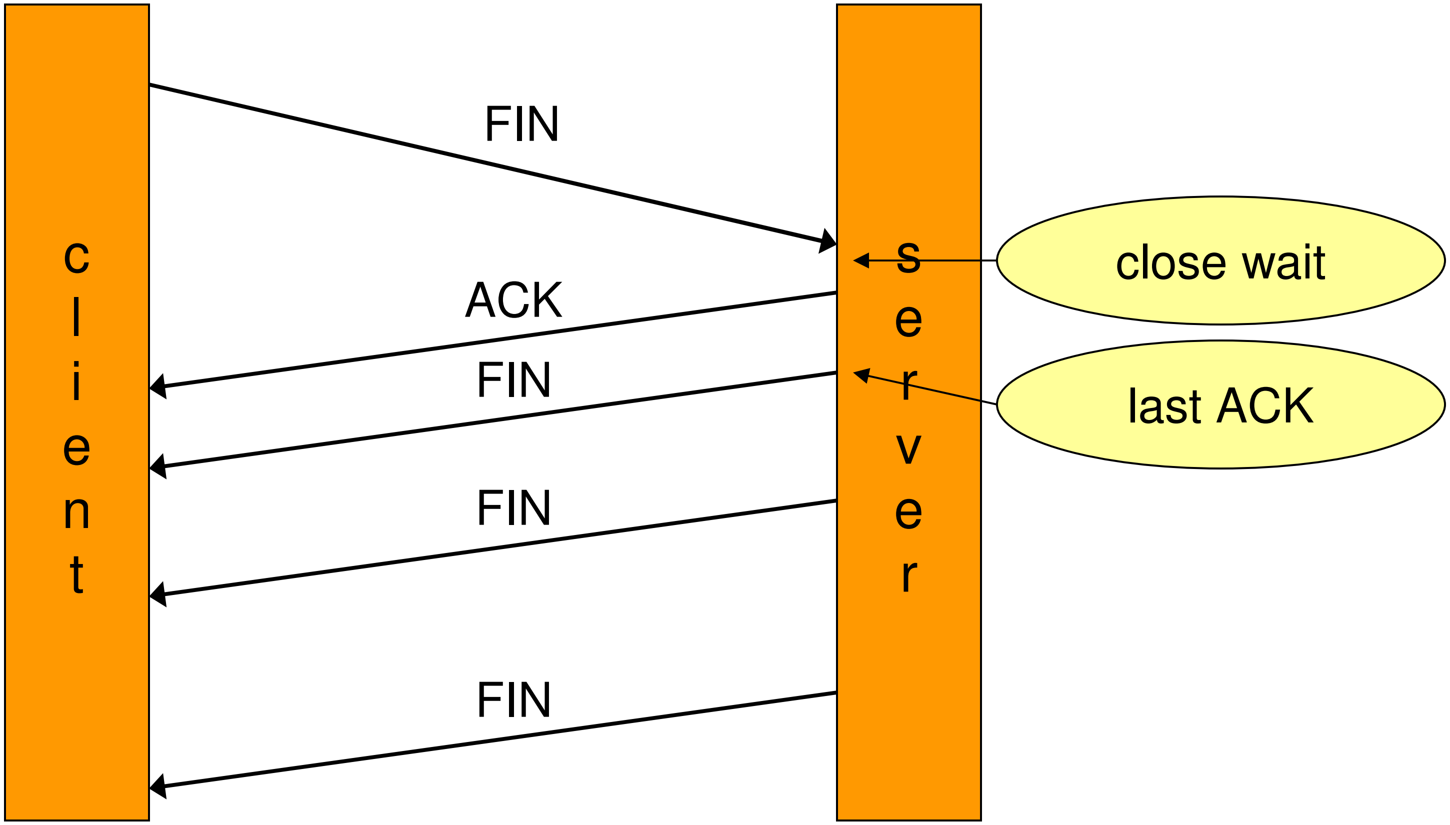
Interrupted 3-way handshake

- Congestion simulated
- Delays between retries and number of retries are measured
- Implementation depended!



Interrupted connection close

- Client sends a close request and simulates congestion
- Delays and number of retries are measured



Examples of responses (Windows)

- Cron-OS database is much smaller then Nmap database
- Results from our laboratory

- Windows XP Professional SP2

```
Cronos_Syn(nbPkt=2%Time=30%p=3000517%p=6011072)
Cronos_LastAck(nbPkt=3%Time=30%Connect=3390%p=2393306%p=480
7053%p=9620439)
```

- Windows 2000 Advanced Server

```
Cronos_Syn(nbPkt=2%Time=30%p=2999441%p=6009839)
Cronos_LastAck(nbPkt=3%Time=30%Connect=2868%p=2985243%p=600
9832%p=12018193)
```

Examples of responses (MacOS)

- FinWait is another test (forcing the server to close the connection)

- MacOS 10.4

```
Cronos_Syn(nbPkt=3%Time=30%p=2793508%p=6004038%p=12001192)
Cronos_FinWait(nbPkt=4%Time=30%Connect=56490%p=402151%p=300
3390%p=5999090%p=12002180)
Cronos_LastAck(nbPkt=4%Time=30%Connect=5305%p=1422542%p=300
5214%p=5999093%p=11998565)
```

Examples of responses (Linux)

- RedHat 9 (Linux kernel 2.4)

  Cronos_Syn(nbPkt=3%Time=30%p=3993350%p=6000931%p=12001043)

  Cronos_LastAck(**nbPkt=7**%Time=30%Connect=4333%p=198878%p=419564%p=839831%p=1680804%p=3360481%p=6720788%p=13440839)

- Mandrake 9.2 (Linux kernel 2.4)

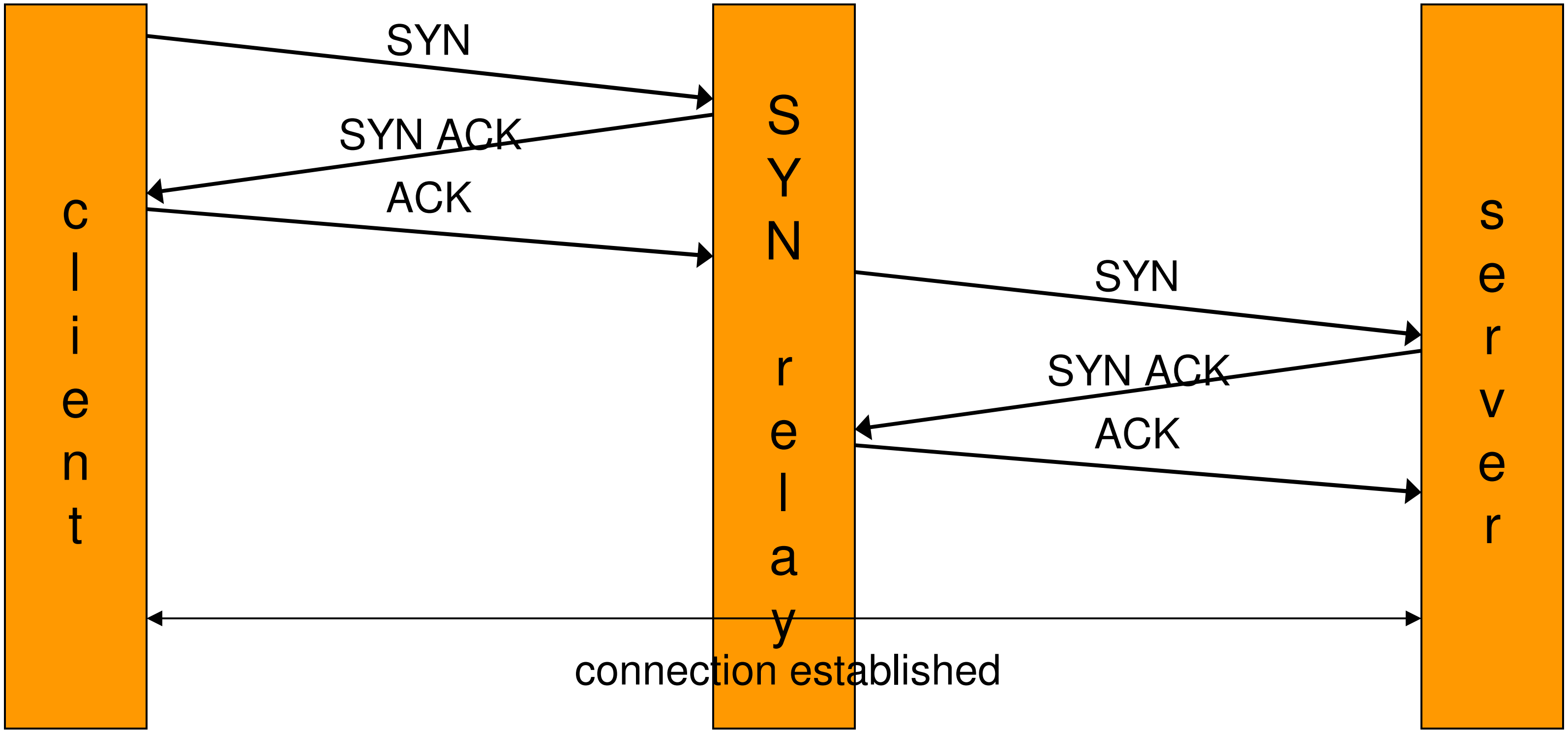
  Cronos_Syn(nbPkt=3%Time=30%p=2996532%p=5980090%p=12203395)

  Cronos_LastAck(**nbPkt=7**%Time=30%Connect=25714%p=203043%p=419576%p=839991%p=1680589%p=3360446%p=6744802%p=13462871)

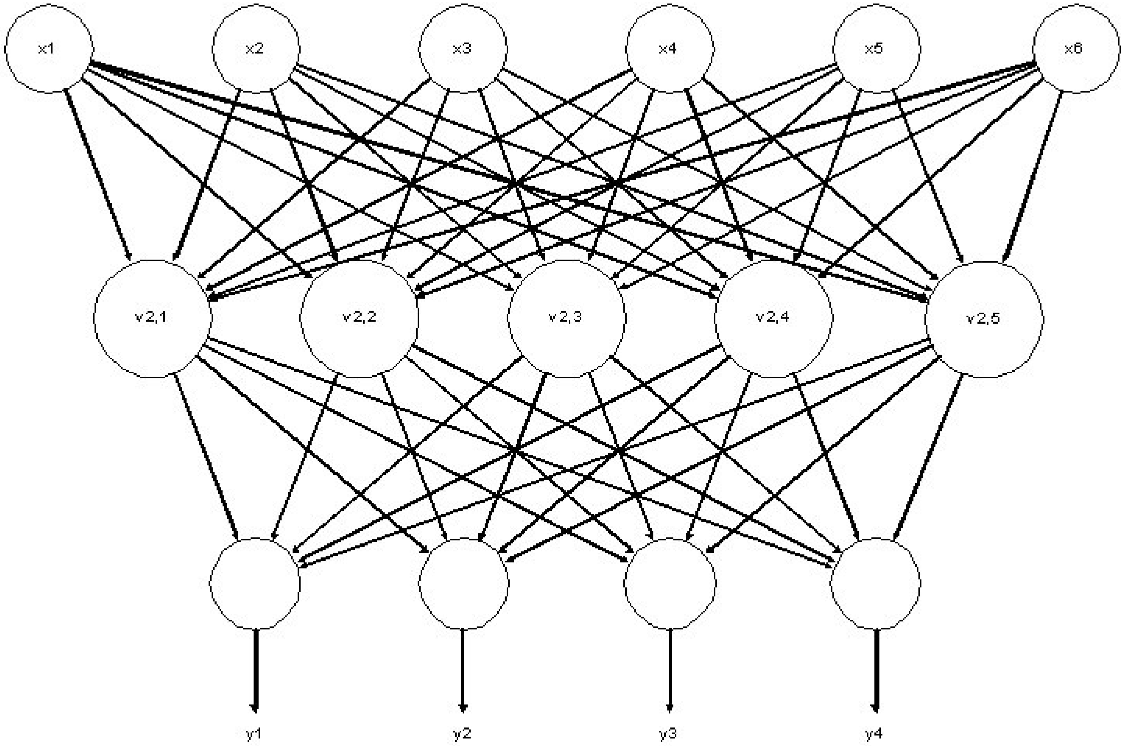
Advantages of Cron-OS

- SYN test sends only 1 packet
- LastAck test sends 3 packets

- Presence of a SYN relay : LastAck test still works



Neural network topology for Cron-OS



input layer : 26 neurons

- delays
- number of retries

hidden layer : 16 neurons

output layer : 6 neurons

Result of the experiment

- Cron-OS + neural networks is usefull for recognizing OS families

OS family	Machines	Match	Mismatch	No match
AIX	1	1	0	0
Linux 2.2	1	1	0	0
Linux 2.4	8	8	0	0
MacOS	3	3	0	0
Solaris	2	0	0	2
Windows	51	51	0	0

Thank you for your attention!

Questions?

Contact information

- For more information about this project:

  http://www.coresecurity.com/corelabs/projects/

- Contact us if you have questions, comments or if you want to look at the source code of the tools we wrote for this research:

  Javier.Burroni at coresecurity com

  Carlos.Sarraute at coresecurity com