NetSpec & Network Performance

Roel Jonkman, Joseph Evans, Victor Frost

http://www.ittc.ukans.edu/netspec

Information & Telecommunication Technology Center
Department of Electrical Engineering & Computer Science
University of Kansas, Lawrence KS
evans@ittc.ukans.edu

DREN Users Conference
6 November 1997
NetSpec Credits

• **Author:**
  • Roelof Jonkman

• **Faculty:**
  • Joseph B. Evans
  • Victor S. Frost
  • Douglas Niehaus

• **Students:**
  • Beng Ong Lee
  • Yulia Wijata
  • Shyam Murthy
  • Anil Gopinath
Outline

• What is NetSpec?
• Motivation, why NetSpec?
• Installation, implementation details
• Controlling endpoints
• Traffic source
• Example script discussion
• Detailed results of several experiments
• Other tools, performance issues...
Testbeds Using NetSpec

The ACTS ATM Internetwork (AAI)

MAGIC Testbed
What is NetSpec?

- Distributed control framework
- Scripting language
- Programmable traffic source
- Custom extendable
What is NetSpec?

- **Distributed control framework**
  - control of multiple endpoints
  - distribution of load
  - distribution of intermediate nodes
  - distribution is user controlled (arbitrary)

- **Scripting language**
  - very C-like
  - simple control hierarchy (recursive descent)
  - transparent to endpoints (leaves)
  - language reflects distribution of nodes.
What is NetSpec?

• **Programmable traffic source**
  • supports TCP and UDP
  • generic traffic sink
  • **programmable full speed source**
  • **programmable burst and queued burst source**
    – WWW, FTP, Telnet, MPEG, Video conference (VBR), Voice (CBR)

• **Custom extendable**
  • ATM call generator
  • CORBA traffic source/sink
  • Data Stream Kernel interface
Motivations, Why NetSpec?

- Other tools
- Distributed control
- Distributed measurements
- Portable
- Optimized
- Scalable
- Integration
- Transparent
Other Throughput Tools

• **Most notably**
  - **ttcp**
    – simple point to point testing tool
    – sufficient for baseline testing, ie. the connection works or not
  - **netperf**
    – point-to-point oriented tool
    – variety of simple traffic types, no random types

• **Problems with existing tools**
  - all tools available lack the capability of controlling multiple connections
  - tools available are suitable for point-to-point connection testing, not for network testing
Distributed Control

- **Distributed Control**
  - single point of control *(netspec)*
  - multiple stages of control
  - multiple endpoints centrally controlled
Portable

• **Platforms currently supported**
  • Digital Unix 4.0, 3.2, 3.0
  • Solaris 2.4 2.5.[01]
  • SunOS 4.1.3
  • Irix 5.3 6.[23]
  • Linux kernel 2.0.25

• **In the works**
  • FreeBSD
    – compiles, has execution problems
  • Windows NT?
Optimized

- Modular code
- Performance critical sections have been profiled (on Digital Unix)
- Hot spots in performance critical sections have been minimized (hot spots are likely within system calls, read and write most notably)
- Non-critical sections have been optimized for
  - readability
  - use
Scalable

• 40..60 controlled connections is common
• 200 controlled connection experiments have been performed
• Some controller specific scalability issues need to be resolved
  • process/machine limitations will be the limit (and already is in most cases)
  • exploit the inherent parallelism that is available within the framework (this will lead to a considerable speedup in most cases)
  • result reports become bulky quickly – compression?
Integration Examples

• Call generator
  • setup and teardown of SVCs

• CORBA
  • performance, similar to the current test daemon

• Kernel Data Stream interface
  • custom interface to kernel data taps

• SNMP
• Control structure uses simple request/response text based control protocol
• Protocol dictates syntactic rules, not semantic rules
  • parameters are syntactically checked at each recursion step
  • control structure passes parameters transparently
  • endpoints interpret parameters
• Endpoint programming parameters independent of the control structure
• Parameter specifications are up to the developer of endpoints
Installation & Implementation

• **Main components**
  - *user interface* - netspec
  - *daemon multiplexer* - netspecd
  - *control daemon* - nscntld

• **“Test” daemons (endpoints)**
  - *test daemon* - nstestd *(TCP/IP, UDP/IP testing)*
  - *tap daemon* - nstapd *(in progress)*
  - *CORBA daemon* - nscorbad
  - *Data Stream daemon* - nsdstrd
Main Components

- **Multiple nested controllers**
- **User interface is text based**
- **netspecd multiplexer serves as front end for NetSpec components, simplifying interfacing and extension (netspec.conf)**
/etc Files

- /etc/services:
  - netspec 42003/tcp # netspec port

- /etc/inetd.conf
  - netspec stream tcp nowait root /usr/local/bin/netspecd netspecd
```
# config file for netspec, netspecd uses this.
#
# format:
#
#service    userid   executable(abs path)   argv[0..x]

cluster    nobody   /usr/local/bin/nscntld   nscntld
parallel   nobody   /usr/local/bin/nscntld   nscntld
serial     nobody   /usr/local/bin/nscntld   nscntld
test       nobody   /usr/local/bin/nstestd   nstestd
dstream    nobody   /usr/local/bin/nsdstrd   nsdstrd
tap         nobody   /usr/local/bin/nstapd   nstapd
corba       nobody   /usr/local/bin/nscorbad  nscorbad
```
Configuration and Script

• Keywords used in netspec.conf denote the keywords by which endpoints are known
• Controller and endpoints are equivalent from a connection perspective, also from a parsing perspective
• netspecd uses the keyword to determine which netspec component to spawn - this is fully transparent (fork() and exec(), or just an exec())
• There is no distinction between connecting to netspecd and a netspec component directly
Controlling Endpoints

- **Parallel**
  - parallel execution of peers
  - nesting allowed

- **Serial**
  - serial execution of peers
  - nesting allowed

- **Cluster**
  - grouping of communicating peers
  - no nesting allowed
Traffic Sources

- Variety of models
  - host maximum rate
  - Constant Bit Rate (CBR) at user level
  - WWW, World Wide Web
  - FTP, File Transfer Protocol
  - telnet
  - MPEG video
  - voice
  - video teleconference
Traffic Sources

• Most common known random distributions
  • uniform
  • Normal
  • log-normal
  • exponential
  • Poisson
  • geometric
  • Pareto
  • gamma
nstestd - Traffic Source/Sink

- **Full speed (or blast in the vernacular)**
  - as fast as process can generate traffic

- **Burst**
  - paced at fairly coarse grain (limited by the fact that it's running as a user level process)
  - a wide variety of distributions, enabling it to mimic various data traffic types commonly in use
  - novel algorithm, slightly more efficient/precise than existing ones (it uses the underlying system calls directly, increases the complexity of the code and adds some caveats)

- **BurstQ**
  - double buffered version of burst, relieves one major caveat of burst
Burst Algorithm

• **Burst**
  - *if the timer expires before the block generator is done with writing the previous block, this algorithm will fail that cycle, and indicate it by counting the failed cycles*
  - *granularity for random periods is about 50 ms to 100 ms, below those values the precision decreases*
  - *granularity for periodic (CBR) types is 10 ms to 20 ms*
**BurstQ Algorithm**

- **BurstQ**
  - added a queue to the previous algorithm, this eliminates the caveat of having to drop cycles if a write is still being served
  - for random (www, ftp, etc.) distributions that might exceed line rate occasionally this is a more reasonable algorithm
Burst Test Terminology

- *Common versus NetSpec terminology*
CBR (Voice) Example

```
cluster {
  test galaga {
    type = burst (blocksize=144, period=18000, duration=1800);
    protocol = tcp (window=65536);
    own = galaga:42000;
    peer = hopper:42000;
  }

  test hopper {
    type = sink (blocksize=144, duration=1800);
    protocol = tcp (window=65536, rcvlowat=8);
    own = hopper:42000;
    peer = galaga:42000;
  }
}
```
## CBR Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocksize</td>
<td>Size of write/read call for each cycle</td>
</tr>
<tr>
<td>period</td>
<td>Inter arrival period (default assumes microseconds)</td>
</tr>
<tr>
<td>duration</td>
<td>Duration of the test (default assumes seconds)</td>
</tr>
<tr>
<td>rcvlowat</td>
<td>Tcp option that controls the low water mark for the Receive side. (This number determines when the Kernel wakes up the process.) Used to avoid reading to small packets, NetSpec uses the first 8 bytes to stuff the length and sequence numbers. (it is by default 8.)</td>
</tr>
</tbody>
</table>
cluster {
    test galaga.atm.tisl.ukans.edu {
        type = burstq (blocksize=ftpItemSize(min=8),
            repeats=ftpNOfItems(min=1),
            period=ftpSessionInterarrival(lambd=0.000001873, min=1000),
            buffer=65536, duration=1140);
        protocol = tcp (window=1048576);
        own = galaga.atm.tisl.ukans.edu:51101;
        peer = nrl.atm.tisl.ukans.edu:51101;
    }

    test nrl.atm.tisl.ukans.edu {
        type = sink (blocksize=65536, duration=1140);
        protocol = tcp (window=1048576, rcvlowat=8);
        own = nrl.atm.tisl.ukans.edu:51101;
        peer = galaga.atm.tisl.ukans.edu:51101;
    }
}

FTP Example
## FTP Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ftpItemSize</code></td>
<td>Fixed distribution, determines the size of each item transferred.</td>
</tr>
<tr>
<td><code>ftpNOfItems</code></td>
<td>Fixed distribution, determines the numbers of items transferred (mget vs. get)</td>
</tr>
<tr>
<td><code>ftpSessionInterarrival</code></td>
<td>Interarrivals of ftp sessions, Exponential distribution, lambda determines the load, the min parameter limits the minimum values this distribution produces. (Some architectures can deal better with low values.)</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td>Sets the upperbound for read and write calls. If the distributions turn out to be bigger than this number the data is segmented in chunks of the size buffer. (Some distributions have extraneous upperbounds, in order not to distort it much, NetSpec segments.)</td>
</tr>
</tbody>
</table>
cluster {
  test hopper.atm.tisl.ukans.edu {
    type = burstq (blocksize=WWWItemSize(min=8, max=104857600, shape=0.40),
    period=WWWRequestInterarrival(lambda=0.000011916, min=1000),
    buffer=65536, duration=1100);
    protocol = tcp (window=1048576);
    own = hopper.atm.tisl.ukans.edu:51201;
    peer = arl.atm.tisl.ukans.edu:51201;
  }

  test arl.atm.tisl.ukans.edu {
    type = sink (blocksize=65536, duration=1100);
    protocol = tcp (window=1048576, rcvlowat=8);
    own = arl.atm.tisl.ukans.edu:51201;
    peer = hopper.atm.tisl.ukans.edu:51201;
  }
}
### WWW Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWWItemSize</td>
<td>Pareto distribution, shape determines the load, higher means higher load. (Pareto = infinite variance.)</td>
</tr>
<tr>
<td>WWWRequestInterarrival</td>
<td>Exponential distribution</td>
</tr>
</tbody>
</table>
MPEG Example

```c
cluster {
        test galaga.atm.tisl.ukans.edu {
                type = burstq (blocksize=videoMPEGFrameSize(min=8),
                                period=33000, buffer=65536, duration=1140);
                protocol = tcp (window=1048576);
                own = galaga.atm.tisl.ukans.edu:51301;
                peer = nrl.atm.tisl.ukans.edu:51301;
        }

        test nrl.atm.tisl.ukans.edu {
                type = sink (blocksize=65536, duration=1140);
                protocol = tcp (window=1048576, rcvlowat=8);
                own = nrl.atm.tisl.ukans.edu:51301;
                peer = galaga.atm.tisl.ukans.edu:51301;
        }
}
```
### MPEG Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>videoMPEGFrameSize</td>
<td>Size of MPEG frames, log-normal distribution, Defaults are set for each of the frames (I, P and B frames) such that it models a 0.5 Mbps MPEG stream.</td>
</tr>
<tr>
<td>period</td>
<td>MPEG is a periodic data stream.</td>
</tr>
</tbody>
</table>
VBR (Teleconference)

```plaintext
cluster {
  test hopper.atm.tisl.ukans.edu {
    type = burstq (blocksize=videoTeleConferenceFrameSize(min=8),
                   period=66000, buffer=65536, duration=1140);
    protocol = tcp (window=1048576);
    own = hopper.atm.tisl.ukans.edu:51401;
    peer = arl.atm.tisl.ukans.edu:51401;
  }

  test arl.atm.tisl.ukans.edu {
    type = sink (blocksize=65536, duration=1140);
    protocol = tcp (window=1048576, rcvlowat=8);
    own = arl.atm.tisl.ukans.edu:51401;
    peer = hopper.atm.tisl.ukans.edu:51401;
  }
}
```
cluster {
    test galaga {
        type = burstq (blocksize=telnetPacketSize,
                        period=telnetPacketInterarrival,
                        buffer=262144);
        protocol = tcp (window=262144);
        own = galaga:42000;
        peer = hopper:42000;
    }
    test hopper {
        type = sink (buffer=262144, duration=900);
        protocol = tcp (window=262144, rcvlowat=8);
        own = hopper:42000;
        peer = galaga:42000;
    }
}
## Telnet Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telnetPacketSize</td>
<td>Size of packets, fixed distribution</td>
</tr>
<tr>
<td>telnetPacketInterarrival</td>
<td>Interarrival times of packets, fixed distribution</td>
</tr>
</tbody>
</table>
Example - ACTS ATM Satellite

```
class {  
    test porky {  
        type = full (blocksize=32768, duration=60, stamps=16000);  
        protocol = tcp (window=10485760);  
        own = 198.119.5.34:42005;  
        peer = 198.119.5.23:42005;  
    }  

    test 198.119.5.23 {  
        type = sink (blocksize=32768, duration=60, stamps=64000);  
        protocol = tcp (window=10485760);  
        peer = 198.119.5.34:42005;  
        own = 198.119.5.23:42005;  
    }  
}
```
Rate versus Time

![Graph showing rates versus time with transmit rate in red and receive rate in green. The graph displays data points fluctuating over time, with a peak around the 20-second mark before settling into a steady state.]
Rate Variation

Rate variation

Transmit rate
Receive rate

Occurrences

Rate (Mb/s)

0 50 100 150 200 250 300 350 400 450

University of Kansas
System Call Duration
Example - Congestion

<table>
<thead>
<tr>
<th></th>
<th>Tx (Mb/s)</th>
<th>Rx (Mb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-C</td>
<td>29.319</td>
<td>29.287</td>
</tr>
<tr>
<td>B-D</td>
<td>29.366</td>
<td>29.204</td>
</tr>
</tbody>
</table>
Aggregate Network Throughput

- **Throughput metrics**
  - maximum losses throughput
  - peak throughput
  - full load throughput
    - transfer from local to remote host memory as fast as possible
Throughput vs. Aggregate Load

Throughput versus offered load

Maximum lossless

Peak

Aggregate throughput (Mb/s)

TCP pacing
No pacing

Aggregate pacing bandwidth (Mb/s)
Example - Large Network

- Large experiment, 10 hosts

* Hosts using small windows

University of Kansas
Example - Many Hosts

- 10 host experiment

<table>
<thead>
<tr>
<th>Host Pair</th>
<th>Individual Throughput (Mb/s)</th>
<th>Simultaneous Throughput (Mb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDC SPARC-10 to KU SPARC-10</td>
<td>23.10</td>
<td>19.12</td>
</tr>
<tr>
<td>TIOC SPARC-10 to KU SPARC-20</td>
<td>15.78</td>
<td>13.09</td>
</tr>
<tr>
<td>ARL SPARC-10 to GSD SPARC-20</td>
<td>11.21</td>
<td>9.60</td>
</tr>
<tr>
<td>CEWES SPARC-10 to EDC SPARC-20</td>
<td>8.89</td>
<td>8.15</td>
</tr>
<tr>
<td>WPAFB SPARC-10 to KU SPARCserver-2000</td>
<td>14.86</td>
<td>5.23</td>
</tr>
</tbody>
</table>
Performance Issues

- Variety of factors
- Highly host (machine and OS) dependent
- Some tweaking required...
Throughput - Factors

- Protocol and application
  - TCP windows
    - window size
    - round-trip time
  - packet size
    - application buffer
    - TCP MTU
    - IP MTU
- error recovery, retransmission strategies
  - fast retransmit
Throughput - Factors

• Host Issues
  • operations
    – memory adapter copy
    – packet processing (checksum, etc.)
    – kernel/user memory copy
  • factors
    – adapter bandwidth
    – I/O bus bandwidth
    – memory bandwidth
    – CPU speed
    – operating system
    – system timers (DEC AXP Digital Unix - 1 ms, SunOS/Solaris - 10 ms)
    – load
Throughput - Factors

- Network issues
  - link speeds
  - bandwidth mismatches
  - competing traffic
  - switch/router buffers
  - congestion control support
    - credit flow control
    - ABR
    - early packet discard
WAN Throughput - Examples

• Protocol and Application
  • TCP windows - window size, round-trip time effects
  • packet size - application buffer, TCP/IP MTU

• Host Issues
  • architecture and operating system
  • packet processing, throughput and PDU counts
  • memory bandwidth

• Network Issues
  • competing traffic
  • bandwidth mismatches
  • switch/router buffers
Throughput - Window Size, Local

- Two DEC AXP-3000/700s at TISL
- Ping times of ~ 1 ms
- Window size not dominant effect

![Throughput vs. TCP Window Size, Buffer = 131072](chart.png)
Throughput - Window Size, WAN

- The good…
  - most modern operating systems support large windows
- The bad…
  - most applications do not exploit large windows
- And the ugly…
  - the defaults are just not good enough
Throughput - Window Size, WAN

- Abandon hope, all ye…
  - expect a particular bandwidth
  - limited by window limits of applications
  - at what ping time does system fail to meet expectations

<table>
<thead>
<tr>
<th>BW Expected</th>
<th>64 kB window RTT (ms)</th>
<th>48 kB window RTT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000,000</td>
<td>0.0524288</td>
<td>0.0393216</td>
</tr>
<tr>
<td>34,000,000</td>
<td>0.015420235</td>
<td>0.011565176</td>
</tr>
<tr>
<td>45,000,000</td>
<td>0.011650844</td>
<td>0.008738133</td>
</tr>
<tr>
<td>100,000,000</td>
<td>0.00524288</td>
<td>0.00393216</td>
</tr>
<tr>
<td>134,000,000</td>
<td>0.003912597</td>
<td>0.002934448</td>
</tr>
<tr>
<td>155,000,000</td>
<td>0.003382503</td>
<td>0.002536877</td>
</tr>
</tbody>
</table>
Throughput - User Buffer Size

- May be limiting component if sufficiently small
- Various architectures differ significantly
Throughput - TCP/IP MTU Size

- DEC AXP-3000/700 with two OC-3c ATM cards configured as a firewall
- Proxy was plug-gw from TIS fwtkV1.3
- TCP/IP MTU size critical
## WAN Throughput - Architecture

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Operating System</th>
<th>Transmit Throughput (Mb/s)</th>
<th>Receive Throughput (Mb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 5000/240</td>
<td>ULTRIX 4.3</td>
<td>54.5</td>
<td>35.8</td>
</tr>
<tr>
<td>Pentium-100</td>
<td>Linux 1.3.91</td>
<td>125.0</td>
<td>70.0</td>
</tr>
<tr>
<td>Pentium-200</td>
<td>Linux 2.0.25</td>
<td>130.0</td>
<td>130.0</td>
</tr>
<tr>
<td>SPARC-10/51</td>
<td>SunOS 4.1.3</td>
<td>62.0</td>
<td>99.0</td>
</tr>
<tr>
<td>SPARC-20/40</td>
<td>Solaris 2.4</td>
<td>83.0</td>
<td>95.0</td>
</tr>
<tr>
<td>SPARC-20/2x125</td>
<td>Solaris 2.4</td>
<td>130.5</td>
<td>115.0</td>
</tr>
<tr>
<td>DEC Alpha 3000/400</td>
<td>OSF/1 v3.0</td>
<td>130.6</td>
<td>134.1</td>
</tr>
<tr>
<td>DEC Alpha 3000/400</td>
<td>Digital Unix T4.0</td>
<td>133.2</td>
<td>133.5</td>
</tr>
<tr>
<td>DEC Alpha 3000/600</td>
<td>OSF/1 v3.2</td>
<td>133.9</td>
<td>133.5</td>
</tr>
</tbody>
</table>
Throughput - MTU Size & Architecture

- Critical fundamental component in throughput limits
- Various architectures differ significantly
Throughput - Packet Processing

- **Processing limits on packets per second**

- **Various architectures differ significantly**
Throughput - Memory Effects

- Critical fundamental component in throughput limits
- Various architectures differ significantly
- Effect of cache evident
Throughput - Congestion Effects

- Three pairs of hosts competing for AAI DS3 link
- WPAFB & NRL to KU, ARL to GSD
- Paced transmit hosts to adjust offered load
- High loss rate (17%) when offered load above link rate

![Throughput versus Offered Load](image)
Throughput - Bandwidth Mismatch

- Rate mismatch at DS3/OC-3c switch
- TCP performance poor
- Why asymmetric?
Throughput - Switch Buffer Size

- Rate mismatch at DS3/OC-3c switch
- NRL switch
  - FORE Rev. C, large buffers
- Sprint GSD
  - FORE Rev. A, small buffers
Throughput - Switch Buffer Size

- Rate mismatch at DS3/OC-3c switch
- NRL switch - FORE Rev. C, large buffers
- Sprint GSD - FORE Rev. A, small buffers
Tools - Data Collection & Plotting

• Ping pages
  • records current state of network
  • historical data also available

• Data collection
  • SNMP queries to switches
  • port, VP, VC statistics

• Data display
  • WWW interface for plotting
  • other visualization tools in progress
Tools - Data Plotter Example

KU AAI Data Plotter

Select a Data Set

- Traffic Flow Out Port B2 on vзвыжин телеграф, net -- ACTS OC3
- Traffic Flow Out Port B2 on vзвыжин телеграф, net -- ACTS OC3
- Traffic Flow Out Port B3 on vзвыжин телеграф, net -- OC3 port connecting to bnta
- Traffic Flow In Port A1 on vзвыжин телеграф, net -- OC3 port connecting to bnta2
- Traffic Flow Out Port A1 on vзвыжин телеграф, net -- OC3 port connecting to KU

X axis

Y axis

View Data Plot

- View many hours? 1
- Show data to nearest 600 sec.

View Specified Time Range (All times in 24-hour notation)

- Start Time: [2004/06/22 13:37:47 [Midnight]]
- End Time: [2004/06/22 14:54:30 [Midnight]]

Click Here to Create Plot: View

*Return to KU AAI Home page

University of Kansas
Tools - Data Plotter Example

- KU switch port to MAGIC and AAI networks
Tools - Cell-Level Tools

- Developed at DEC Systems Research Center
  - author: Hal Murray
- Based on DEC ATM OC-3c (OTTO) interfaces
- Measure fine-grain delays
- Study cell-level CDV in large networks
Delays - Histograms, KU/TIOC
Yet More Tools...

• netstat
  • interface statistics
  • protocol statistics (tcp_stats script)
• traceroute
• atmstat
• ATM signaling traces
• ATM route tracing...
Summary

• **NetSpec**
  • distributed control framework
  • scripting language
  • programmable traffic source
  • custom extendable

• **Other tools**

• **Critical performance factors**
For More Information...

• AAI Web sites
  • NetSpec
    http://www.ittc.ukans.edu/netspec
  • AAI home page
    http://www.arl.mil/AAI/
  • KU AAI home page
    http://www.ittc.ukans.edu/aai/
  • KU data plotter
    http://www.ittc.ukans.edu/cgi-bin/aai/plot_select.pl

• MAGIC Web sites
  • MAGIC home page
    http://www.magic.net/