Testing Command & Control Systems: A Recipe for Success

Mr. Chris E. Johnson

SPAWAR Systems Center, San Diego

San Diego, CA 92152

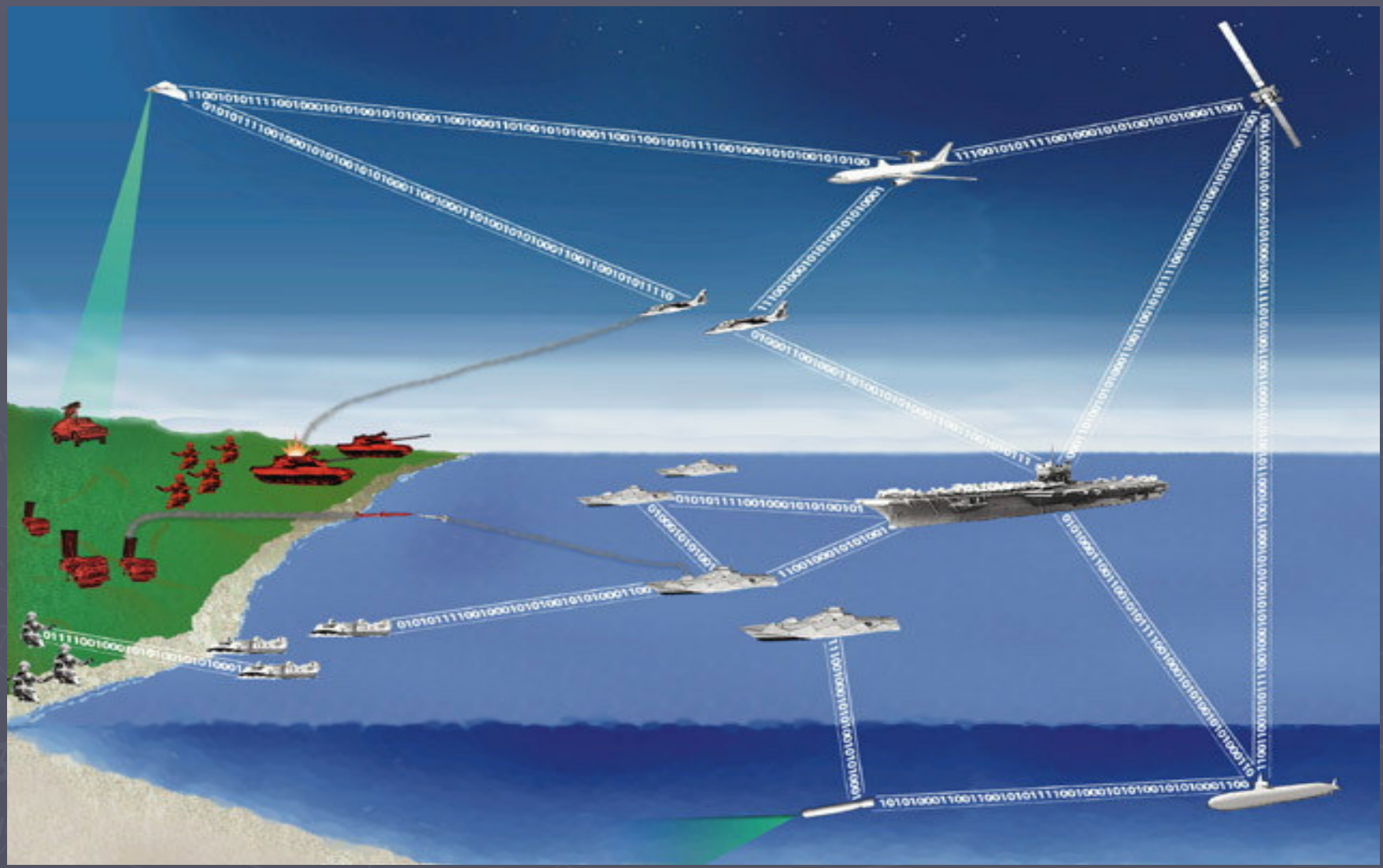
Overview

- Characterize a Command and Control (C2) system
- Understand the decisions involved in testing a "near real time" system
- Recipe for conducting a performance test
- Data Analysis

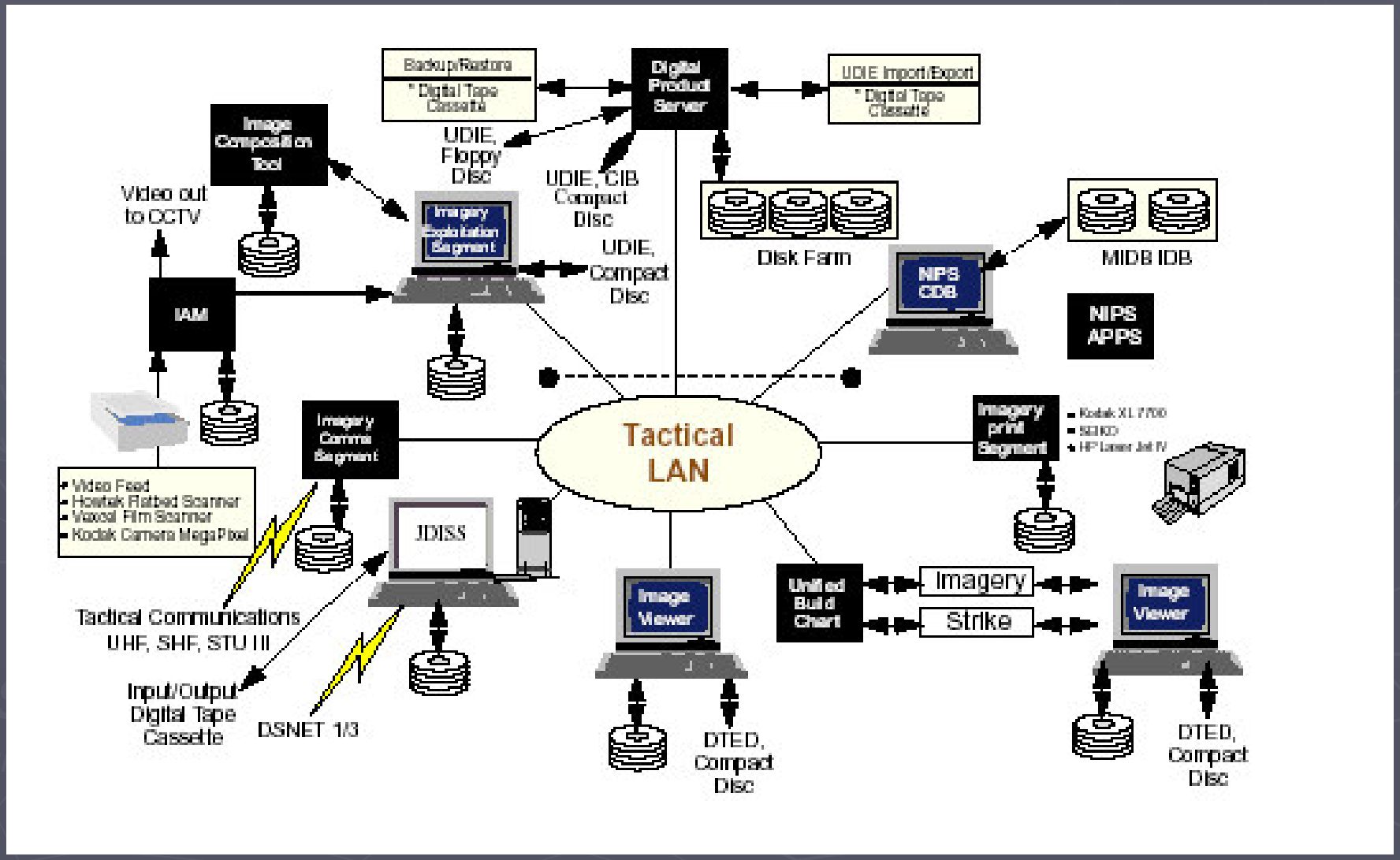
This is the problem…

- Conduct performance testing on a "near real-time system"
  - Thousands of participating servers and clients
  - Geographic displacement
  - High data rate generating sensors
  - Operating in a "quasi-Internet" environment
  - "Reasonable performance" expectations

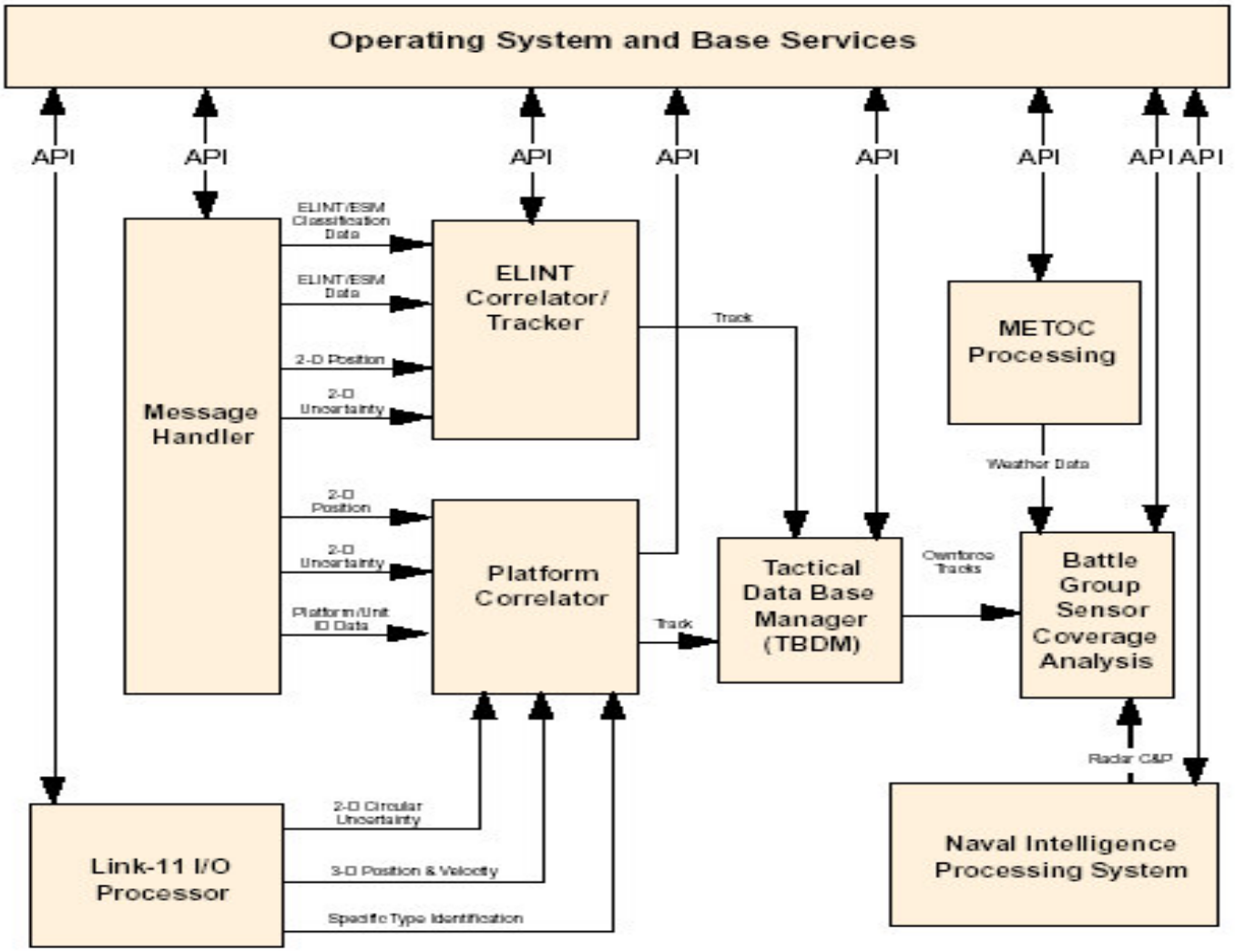
What is Command and Control?



What a C2 System Looks Like…



Inside the Software of a Navy C2 System…



Software/Hardware Specs

Item	Characteristics
Platform	Sun Ultra's 60 and 80 Sun Netras 1405 Sun Blade 2000 Dell and Intel P4 (Desktop and Laptop)
Software	8.2M SLOC Expandable via additional applications to 17.3M LOC written originally in C and migrated to Java
Network	ATM Backbone using Xylan Smart Switch with Cisco Routers
Bandwidth	Simulate range of connectivity's (e.g., airplanes, ships, command centers, hummers, troops in field).
Data Injection Rate	Avg. ~300 updates per minute incoming to server = ~600-800 data packets broadcast via UDP from server to clients over network

Test Purpose

- Primary purpose: Establish a baseline of system performance in a representative architecture with high tempo events occurring
- Secondary purpose: Conduct an in-depth test of performance characteristics with the ability to pinpoint and identify problematic processes

Performance Testing: The Recipe Ingredients

- **System Latency: What generates the highest latency within the system?**
- **User Tasks: While conducting normal user tasks, which ones degrade system performance in our environment?**
- **Data Rates: Where do we target our monitoring efforts: network, client, or server?**
- **Tools: What tools do we use to monitor the test?**
- **Tasks and Data: How do we target specific events and data that generate degradation?**
- **Methods and Architecture: How do we mimic a complex environment with minimal machines?**

What generates the highest latency?

- COMMON:
  - Events that require extended service from the CPU (the larger the process, the more service required)
  - Events that write to I/O
  - Events that "thrash" memory
- SPECIFIC:
  - Redundant calls on Java VM
  - More "Real Time" data produces greater latency
  - Large data volume, with small bandwidth (cramming 10 lbs of stuff in a 5 lb bag!)

While conducting normal user tasks, which ones degrade system performance in our environment?

- Moving large data volumes from server to client
- High broadcast rates of server to other servers and clients based on turning data feeds on and off
- Any user task that would require the system to launch more than approx. 3 processes / applications simultaneously

Where do we target our monitoring efforts: network, client, or server?

- Since we were unsure when we started, we monitored it all and refined as we became experienced
- Based on our results to date, recommend targeting
  - Highest grade and lowest grade servers
  - Highest grade and lowest grade clients
  - Network at both ends of the incoming server
    - Data coming in and data being broadcast through to clients

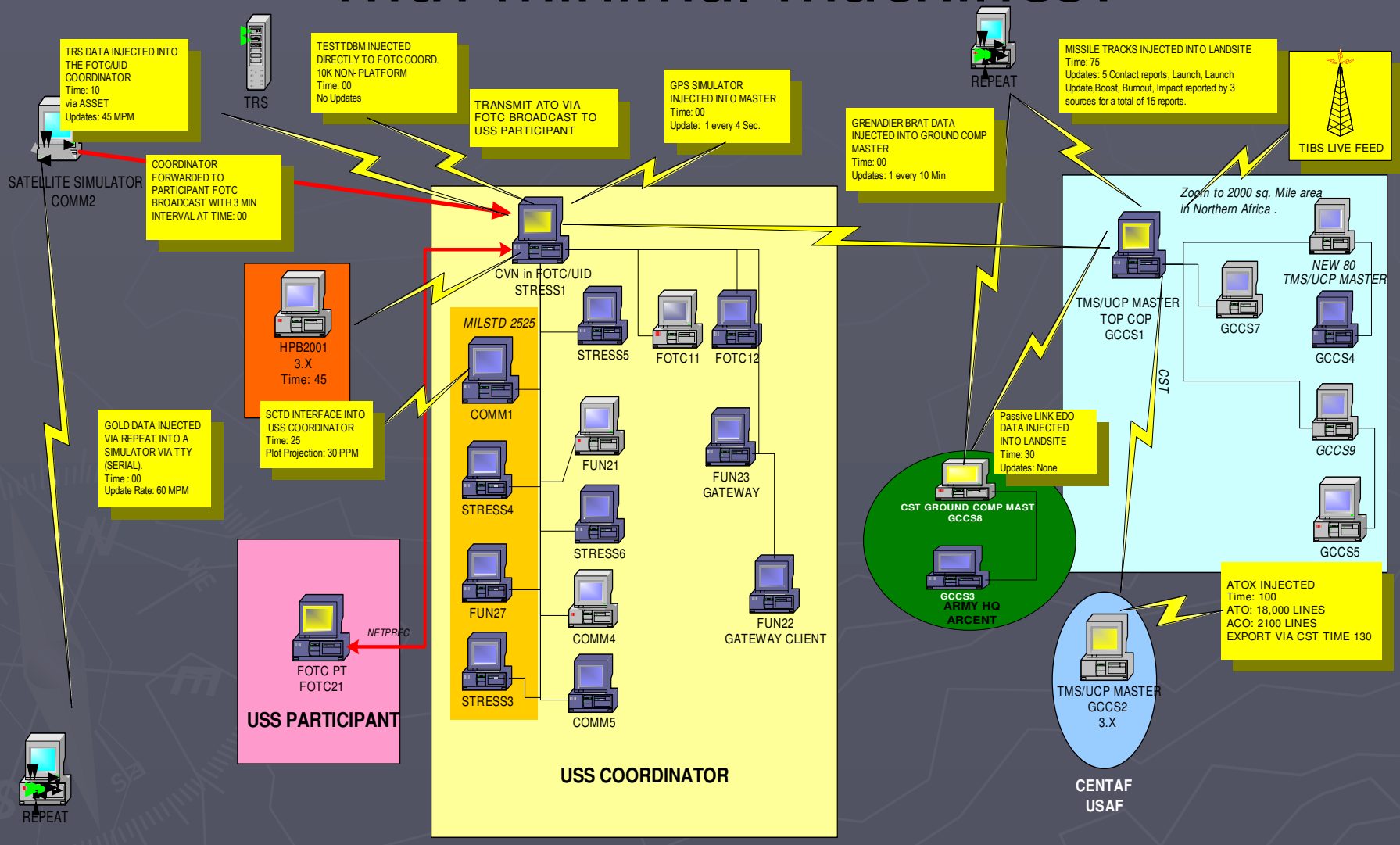
What tools do we use to monitor the test?

- Criteria for tool selection
  - Reside on any platform (Sun, Windows, or HP)
  - Leave a minimal footprint
    - Less than 1% of RAM required while running
    - Less than 1% of CPU service on system under test
  - Provide graphical data for analysis
- Our choice: TeamQuest

How do we target specific events and data that generate degradation?

- Step 1:
  - We ran our smoke test using Team Quest with ~255 user functions
  - We incremented, then decremented our data injects
  - We monitored activity and targeted items that we suspected were problems
- Step 2:
  - We refined our list to 70 user functions which we felt best captured the above items
- Step 3:
  - We conducted multiple dry runs to refine and confirm capability

How do we mimic a complex environment with minimal machines?



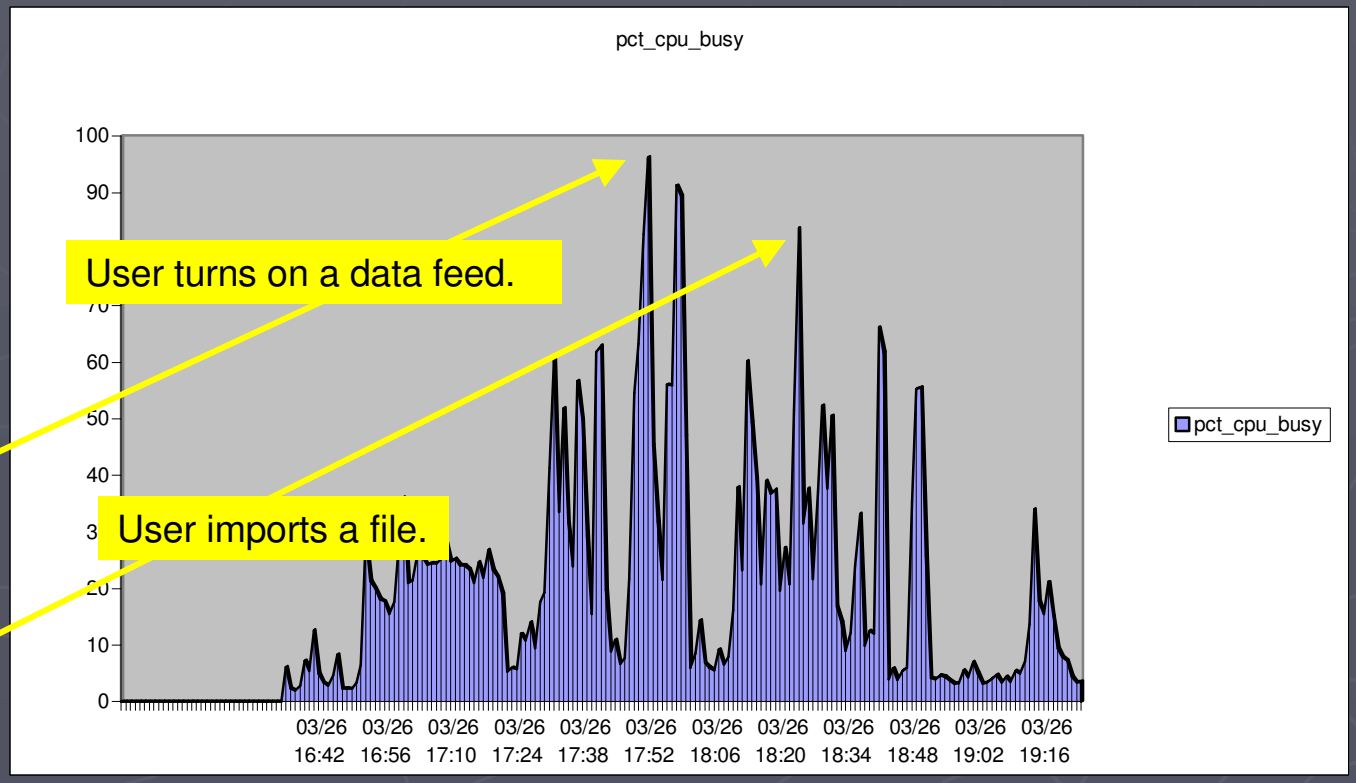
MIXING INSTRUCTIONS: How to generate a repeatable test that allows data to be analyzed correctly

- Base load the machines with predictable data
- Ensure the performance monitor tool is loaded on targeted platforms
- Begin the test with minimal data injection from external sources
- Incrementally add users at specific times
  - Users conduct the exact same script at offset times
  - This step allows for pinpointing tasks that may cause system degradation
- Add data feeds at targeted times
- Increase data injection rates to established targeted amounts
- Collect data and analyze
- …Let Simmer!

Analyzing the Data (Is your test tool your friend?)

Actual Server used during testing 3/26/2001.

03/26 17:34	51.8583
03/26 17:35	31.575
03/26 17:36	23.9583
03/26 17:37	56.6833
03/26 17:38	49.9
03/26 17:39	29.3
03/26 17:40	15.5333
03/26 17:41	61.85
03/26 17:42	63.0083
03/26 17:43	19.9
03/26 17:44	8.95
03/26 17:45	10.875
03/26 17:46	6.66556
03/26 17:47	7.69167
03/26 17:48	21.6333
03/26 17:49	54.5333
03/26 17:50	63.3667
03/26 17:51	82.6667
03/26 17:52	96.2
03/26 17:53	45.4583
03/26 17:54	31.8417
03/26 17:55	21.5667
03/26 17:56	56
03/26 17:57	55.875
03/26 17:58	91.2612
03/26 17:59	89.3952
03/26 18:00	47.0333
03/26 18:01	6.01667
03/26 18:02	8.55833
03/26 18:03	14.4417
03/26 18:04	6.83333



Analyzing the Data

Same Server different graph showing process generating high CPU usage. (Command line statement)

gccs1 SOLARIS:Proc	Mar 26, 2002 17:52:00 - 11:151 of 151					
command	pctcpu	fullcmd	login	nproc	totcpu	rss
<Multi>	95.62	<Multi>	<Mult	151	115	1304088
java	75.96	/h/COTS/JAVA2/bin/../bin/sparc/native_threads/java -Dafw -Xss1	tester	1	91.2	173912
Cartographer	5	Cartographer	tester	1	6	22968
Xsun	3.81	/usr/openwin/bin/Xsun :0 -nobanner -auth /var/dt/A:0-5saydb	root	1	4.57	45536
Tdbm	1.95	Tdbm	root	1	2.34	55168
CSTIeDec	1.48	CSTIeDec CST3X	root	1	1.78	10624
CSTTCP	1.36	CSTTCP CSTTCP	root	1	1.64	13800
java	0.68	/h/COTS/JAVA2/bin/../bin/sparc/native_threads/java -Xss768k -X	tester	1	0.81	48632
dtwm	0.67	dtwm	tester	1	0.81	7520
CSTTCP	0.65	CSTTCP CST3X	root	1	0.79	11040

Summary

- Characterize a C2 system
- Understand the decisions involved in testing a "near real time" system
- Recipe for conducting a performance test
- Data Analysis (The tool can be your best friend!)