Intelligent Networks For Fault Tolerance in Real-Time Distributed Systems

Jayakrishnan Nair

Real Time Distributed Systems

- A Distributed System may follow a traditional Master-Slave Approach for Task Allocation

- A Real-Time DS would have hard real-time constraints for completing each set of Tasks

- Normally such systems are implemented with multiple workstations connected together by a high bandwidth Gigabit network like Myrinet

Fault Tolerance in RTDS

- Many Contemporary Science Applications run as RTDS in fault-vulnerable ambiences
  - Ability to survive faults is required to achieve efficient system throughput and output integrity

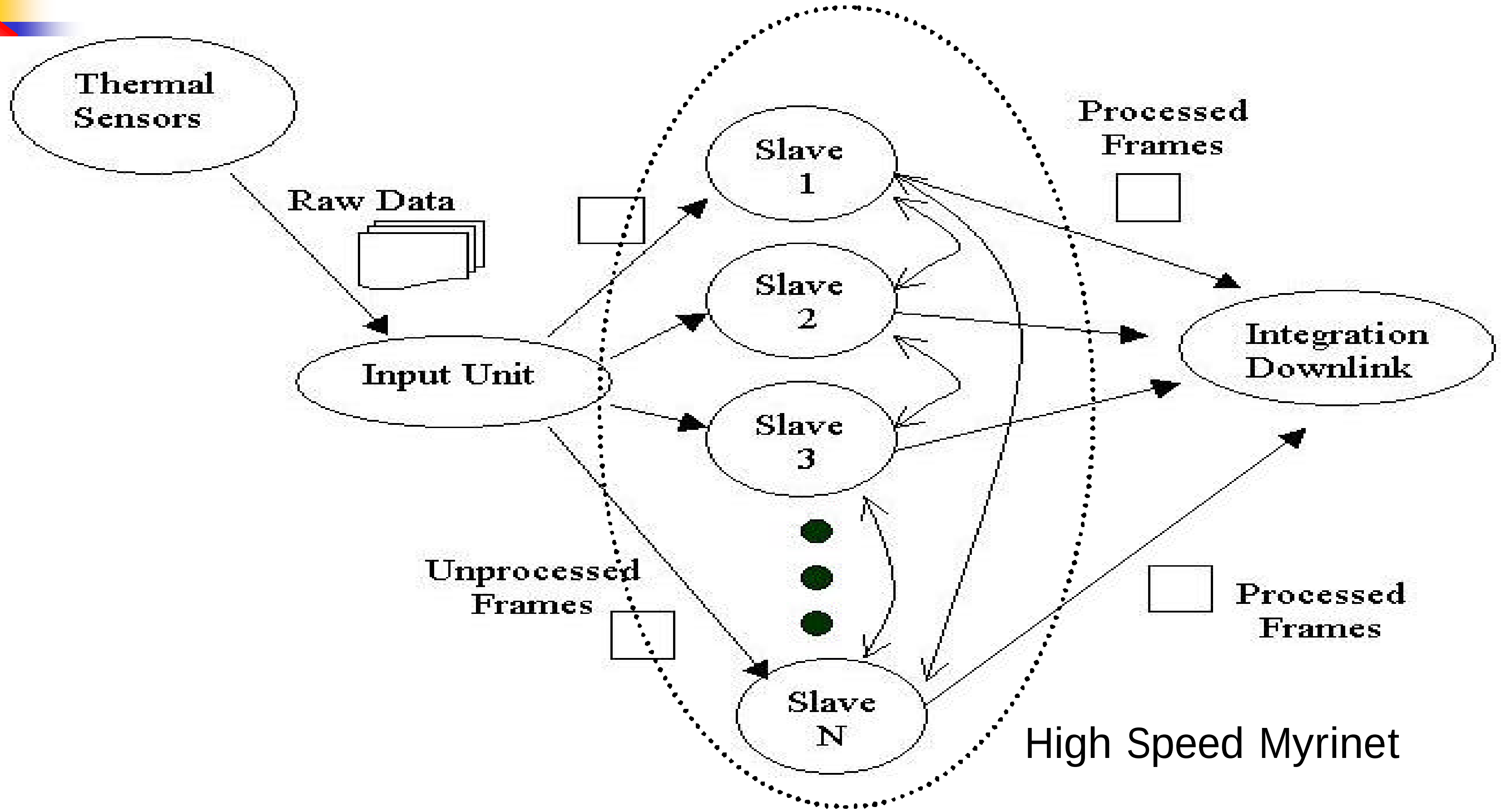
- Space applications run onboard the spacecrafts process huge volumes of Data in real-time

- Raw Data susceptible to bitflips at source due to Charged Particles and Cosmic Rays

A Benchmark Application.

- Orbital Thermal Imaging Spectrometer (OTIS)
  - Orbiting probe-based distributed software
  - Collects radiation data from the atmosphere using onboard sensors
  - Processes it onboard to obtain temperature and emissivity mappings of the geographical location
- Susceptible to Data Faults
  - Bombardment by free-moving charged particles (alpha)
  - South Atlantic Anomaly

OTIS System Architecture



Input Unit periodically sends the unprocessed FITS frames to the slaves over Myrinet for real time processing

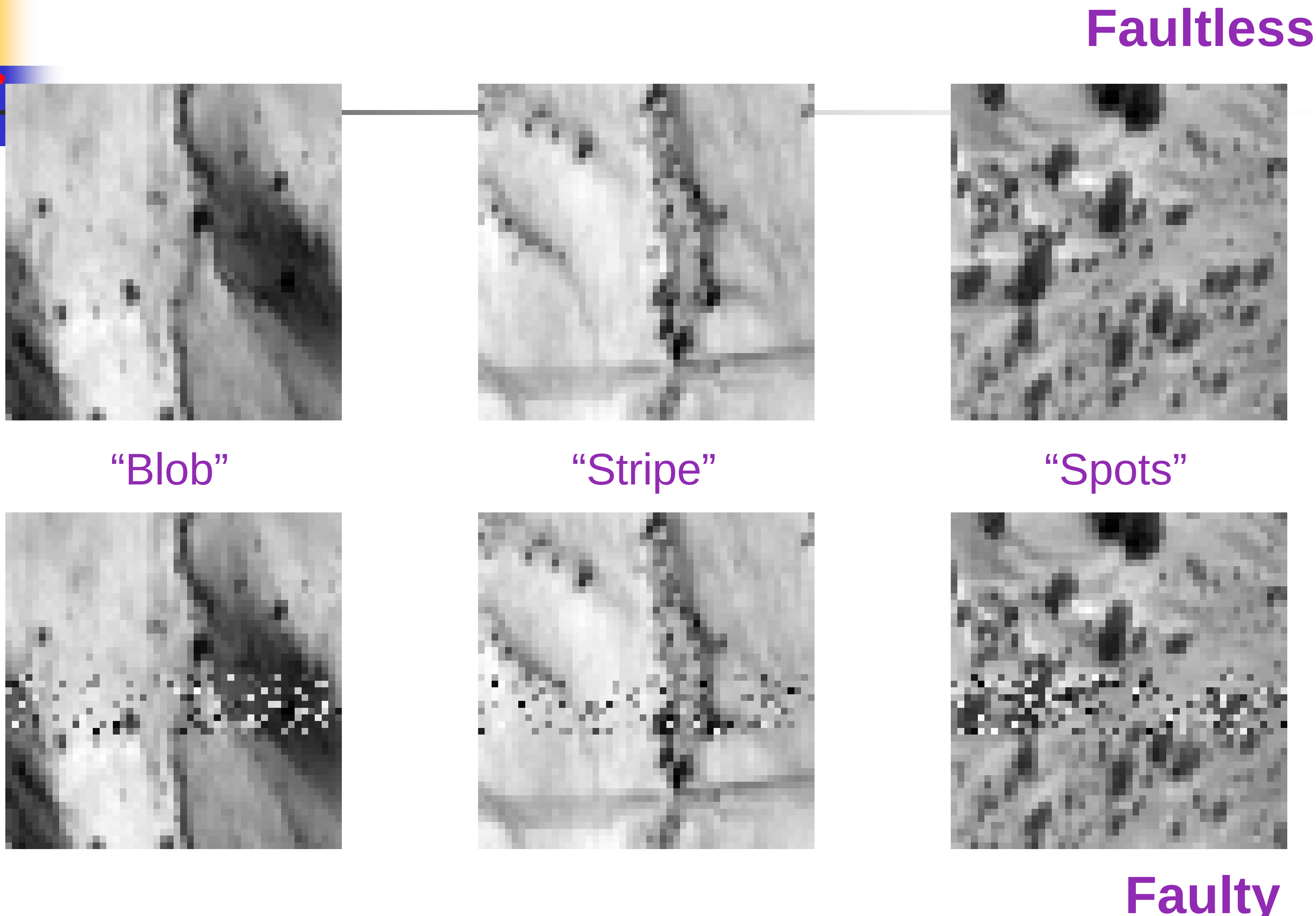
Effects of Faults

- Drastic reduction in the reliability of output
  - Hence the science information garnered is less
  - Lower Accuracy for Weather Prediction & Analysis

- Abnormal Process Terminations and Node Hangs
  - Data Faults can lead to invalid states in the FSM of the processing applications

- Discarding the input for fresh-set of observations not feasible in real-time

OTIS Datasets

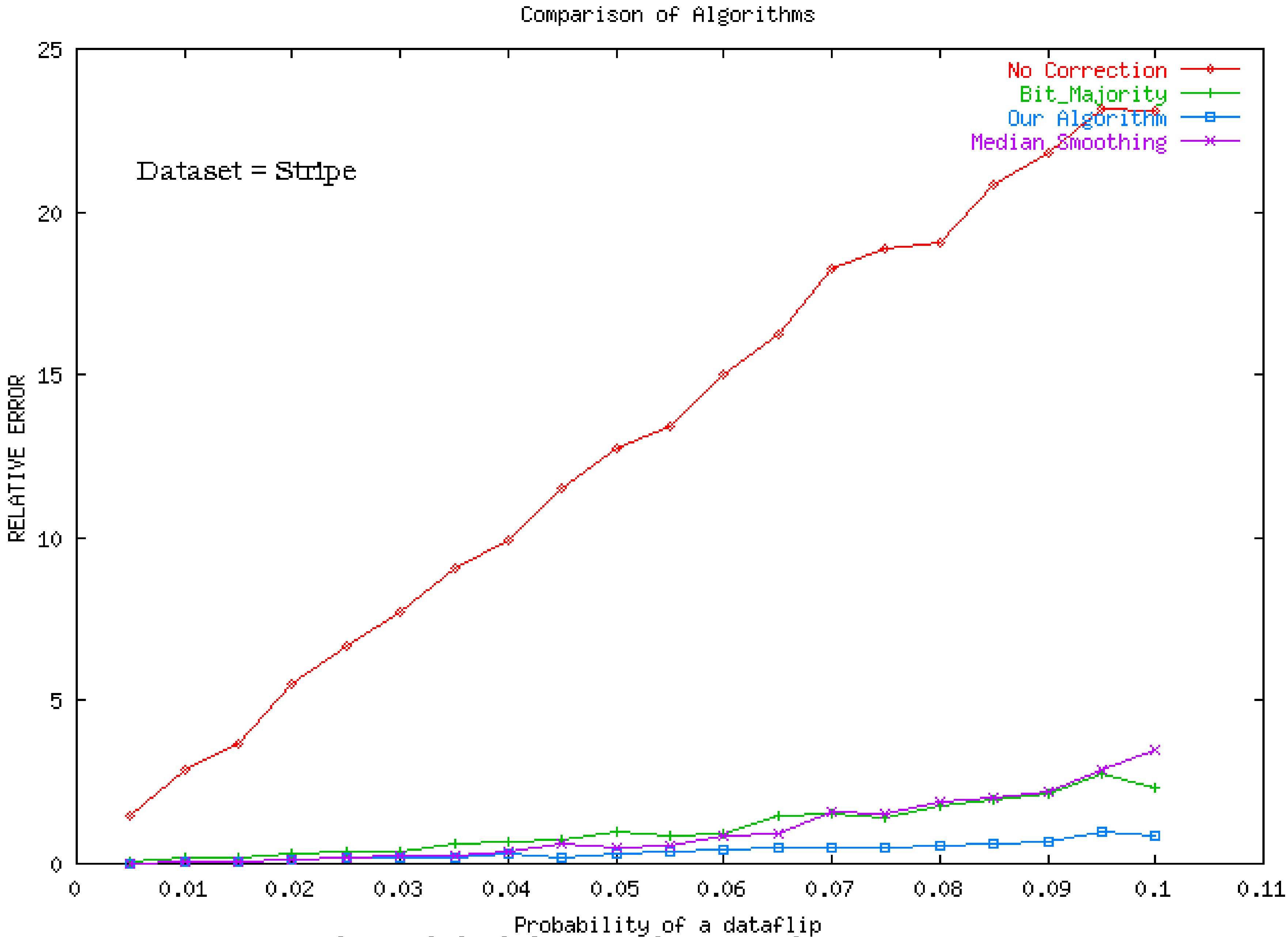


Input Preprocessing

- No inherent error-correction available at source
- Use Data Redundancy, Correlation and Application Semantics to Identify Data Faults
- Preprocessing Algorithms do Dynamic Statistical Analysis of Input and identify corrupted bits
  - Significantly reduced average error in the datasets
  - Input Integrity Assurance to the System

Sample Faulty Data

Pixel Number	Original Data from Detector (Unaffected with faults)		Actual data in memory (Affected with faults)	
D1	1 0 1 0 1 1 0 0	172	1 0 1 0 1 1 0 0	172
D2	1 0 1 0 1 1 1 0	174	1 0 1 0 1 1 1 0	174
D3	1 0 1 0 0 1 1 1	167	1 0 1 0 0 1 1 1	167
D4	1 0 1 0 1 1 0 1	173	0 0 1 0 1 1 0 1	45
D5	1 0 1 0 1 1 0 1	173	1 0 1 0 1 1 0 1	173
D6	1 0 1 0 0 1 1 1	166	1 0 1 0 0 1 1 1	166
D7	1 0 1 0 1 1 1 1	175	1 1 1 0 1 1 1 1	239
D8	1 0 1 0 1 1 1 1	175	1 0 1 0 1 1 1 1	175



Preprocessing highly reduces input error

Challenges in Implementation

- Though Preprocessing is highly effective, it has implementation ramifications in a RTDS

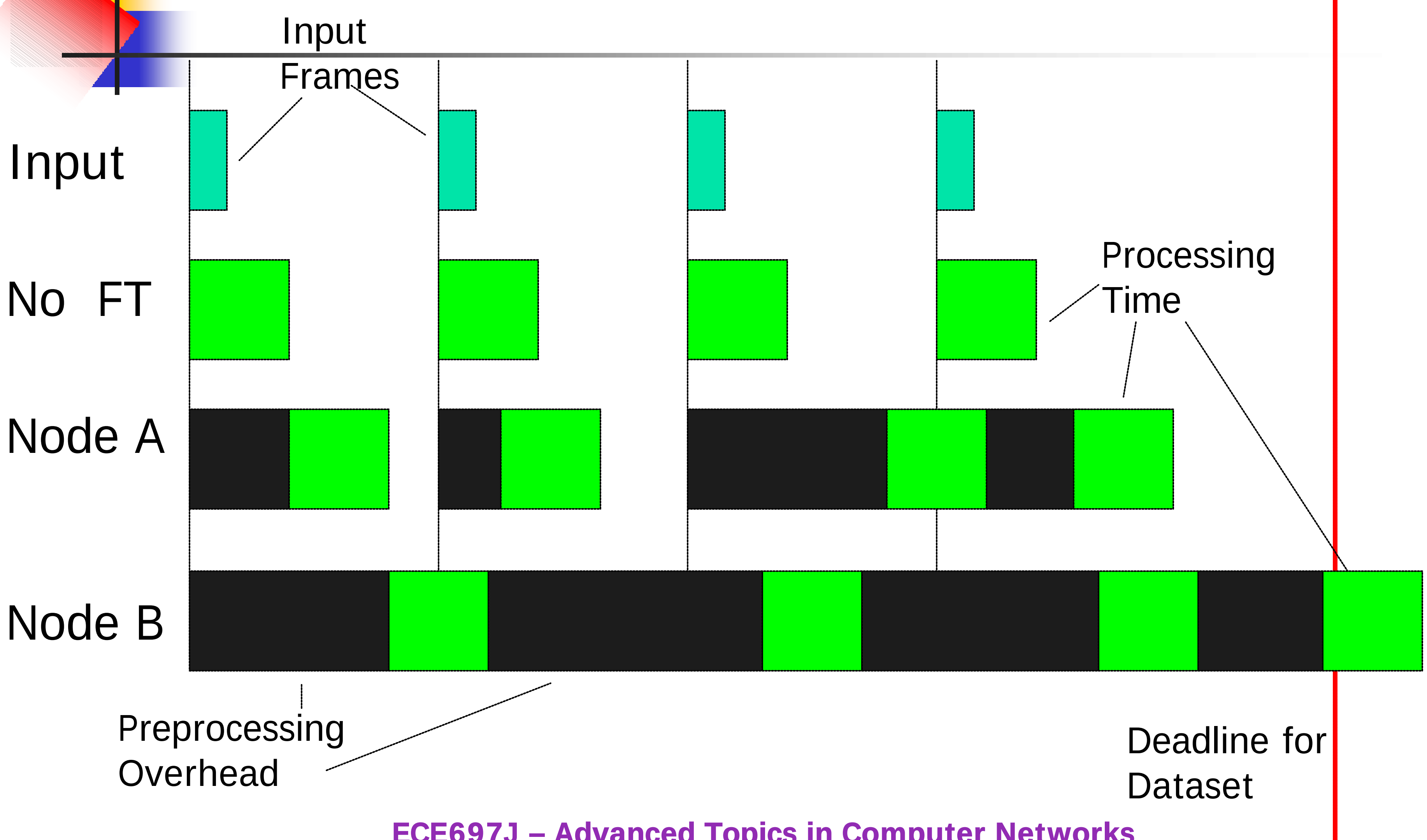
- The overhead due to preprocessing is statically unpredictable, and hence some nodes may fall below others in time – loss of sync!

- If the nodes are naively scheduled, then the accumulated skew due to the preprocessing overhead can eventually cause a deadline miss

Execution Time Varies

The execution overhead due to Preprocessing depends on sensitivity (a dynamic parameter) and the turbulence in data

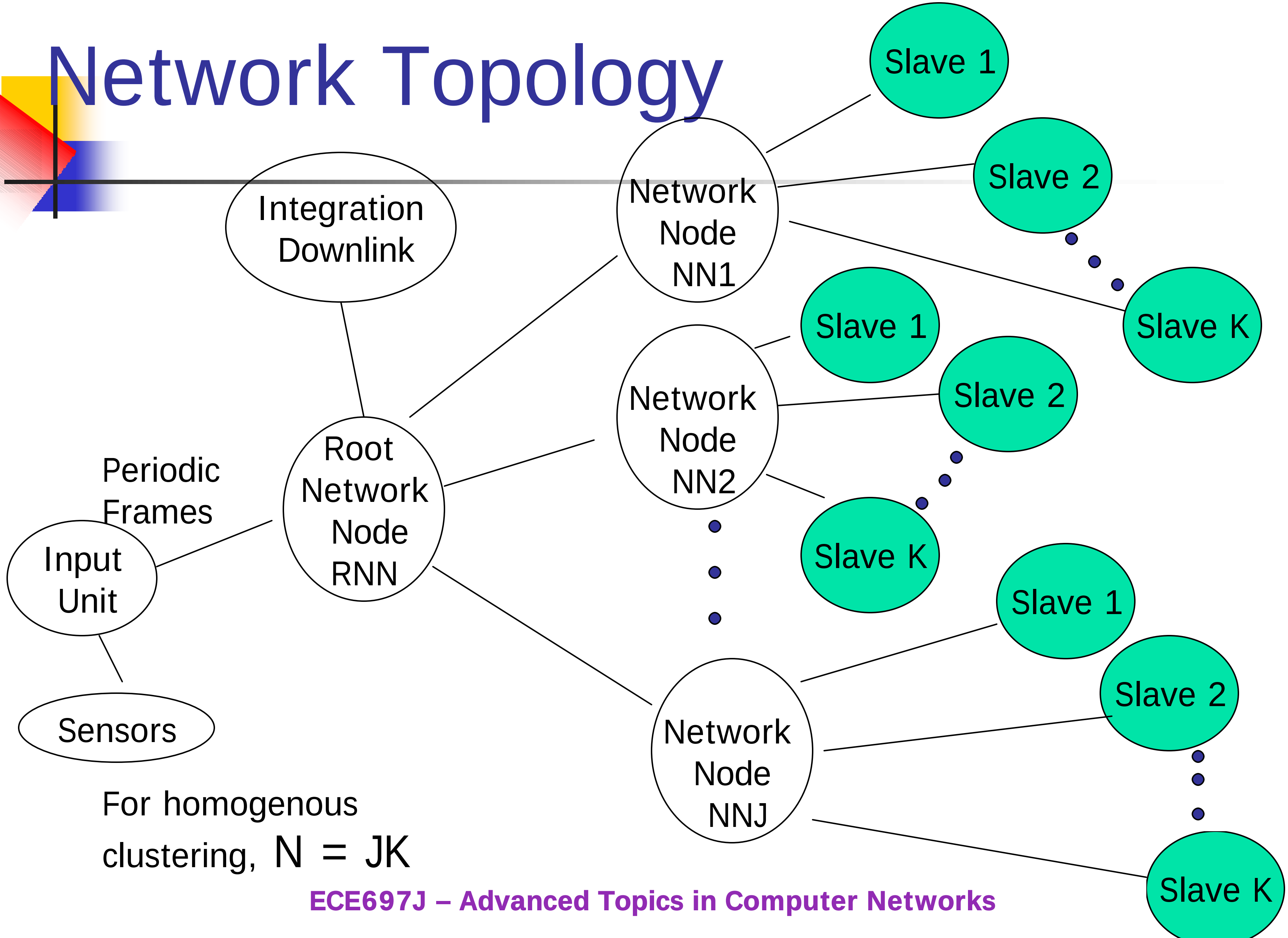
Overhead causing Deadline Miss



Envisaged Solution?

- Network Nodes must intelligently schedule the frames to minimize skew accumulation at slaves
- Statically impossible – hence the network must have do dynamic run-time estimation
- Must keep a lookup table on the skews and pending workloads of each slave [Skew Accumulation Matrix (SAM)]

Network Topology



For homogenous clustering, N = JK

Solution – Intelligent Networks

- The Input Unit is Oblivious to the scheme and just pumps out data frames periodically to RNN
- The RNN allots the frames in a round robin fashion to NNs
- Each NN has a local copy of SAM [Skew Accumulation Matrix]
  - SAM has the hitherto accumulated Skew of all N slaves
  - Initially, SAM has zero value for all the slaves

Solution Scheme (Continued)

- The NN in charge of the current frame does a statistical pre-analysis of the frame for run-time estimation of the preprocessing algorithm
  - Computes the parameters Window-Width (Υ) and Sensitivity (Λ) for the data frame
  - Estimated run-time E is $O(N[\Upsilon^2 + \Lambda])$
  - Computing Υ and Λ for a data frame requires O(1) time
- The NN then looks up the SAM, finds the slave with the lowest accumulated skew (S), and allots the current frame to it, after adding the computed E to its field in SAM
- All the NNs are then updated with the new SAM

An Example

- Let us consider the current SAM as

SAM[] = {35, 45, 70, 54, 33, 57, 49, 51, 54, 47, 38, 42}

- The NN in charge of the current frame computes its Υ and Λ and estimates the run-time as, say, 7.
- The slave with the current lowest S, slave 5 (33) is selected and the frame is dispatched to it for processing (through to the cluster that has it)
- The SAM is updated to all NNs as

SAM[] = {35, 45, 70, 54, 40, 57, 49, 51, 54, 47, 38, 42}

Experimental Framework

- The Simulation Setup has been implemented in a uni-processor system
  - Implementing a real distributed system connected with active network nodes is outside the scope of this project

- Concurrent processes simulate the independent entities like Input Unit, RNN, NNs, Slaves etc.

- Communication between entities in the target network is achieved through inter-process communication.
  - As network latencies in the target network Myrinet are negligible, this model approximates well.

Concurrent Processes

- Input Unit (IU) Process:
  - Reads OTIS data (obtained from the REE project team, NASA) from files (substitute for sensors)
  - Injects Faults (to simulate the vulnerable ambience) using a Fault Injector that randomly flips the bits in data based on a given probability P (uses a pseudo-random-number generator)
  - Periodically sends data to the RNN every 50ms.

- RNN Process:
  - Allots the data frames from the IU to the NN processes in round robin
  - The simulated system has J=3 NNs and K=4 slaves for each NN, hence N = 12

Concurrent Processes (Continued)

- NN Process:
  - For the data frame received, computes the value of the algorithm parameters ϒ and Λ and empirically estimates the runtime E
  - Finds the Slave Q with lowest S from the SAM
  - Increments it's S with E
  - Updates the local copy of the SAM of every NN
  - Sends the Data Frame to the process simulating the slave Q.

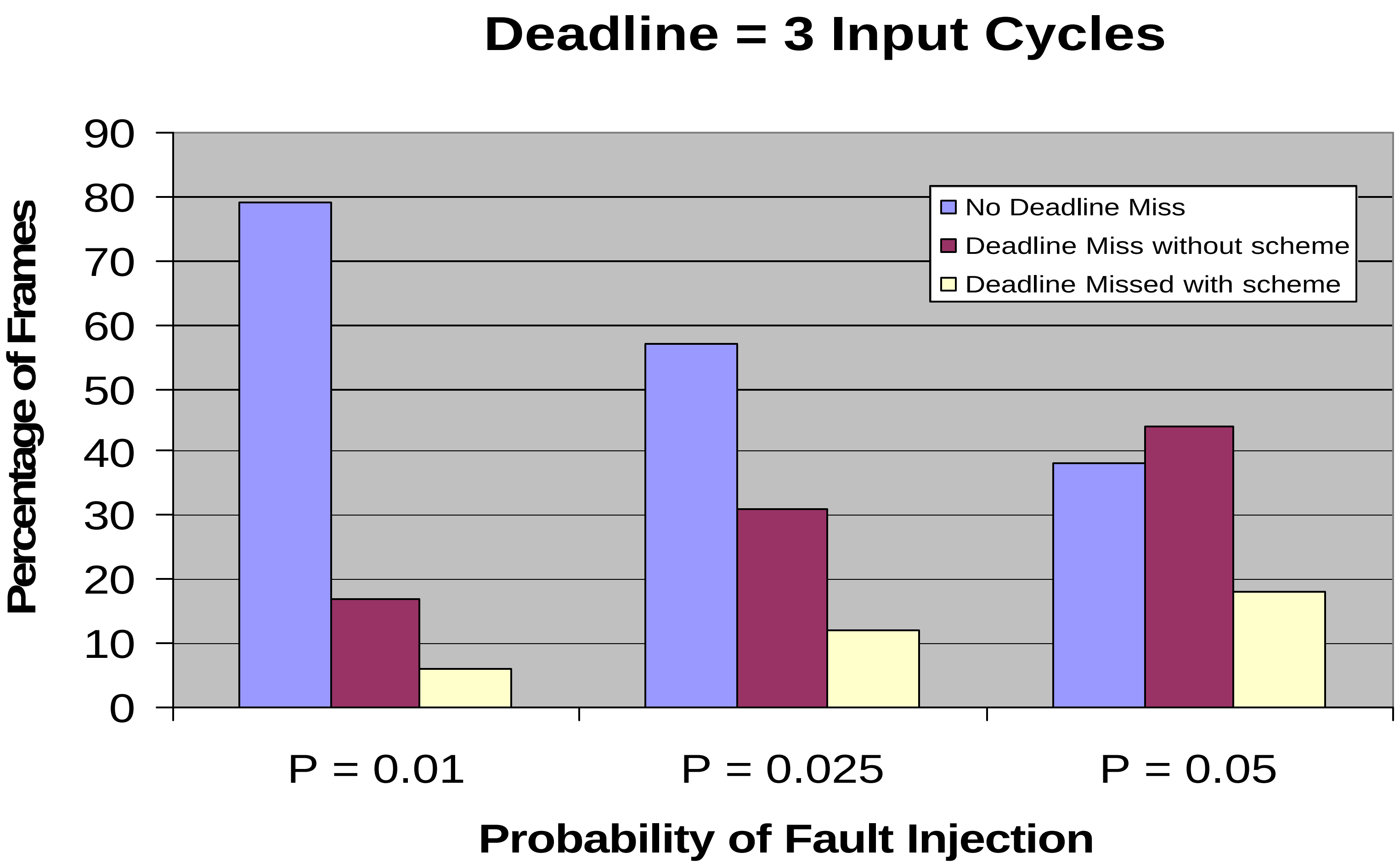
- Slave Process:
  - Receives the Data Frame from NN
  - Preprocesses the data using input preprocessing algorithm
  - Corrects the bitflips identified
  - Processes the cleaned Data to get the OTIS output frame
  - Sends processed frame with seq. no. to the Integration Process

Concurrent Processes (Continued)

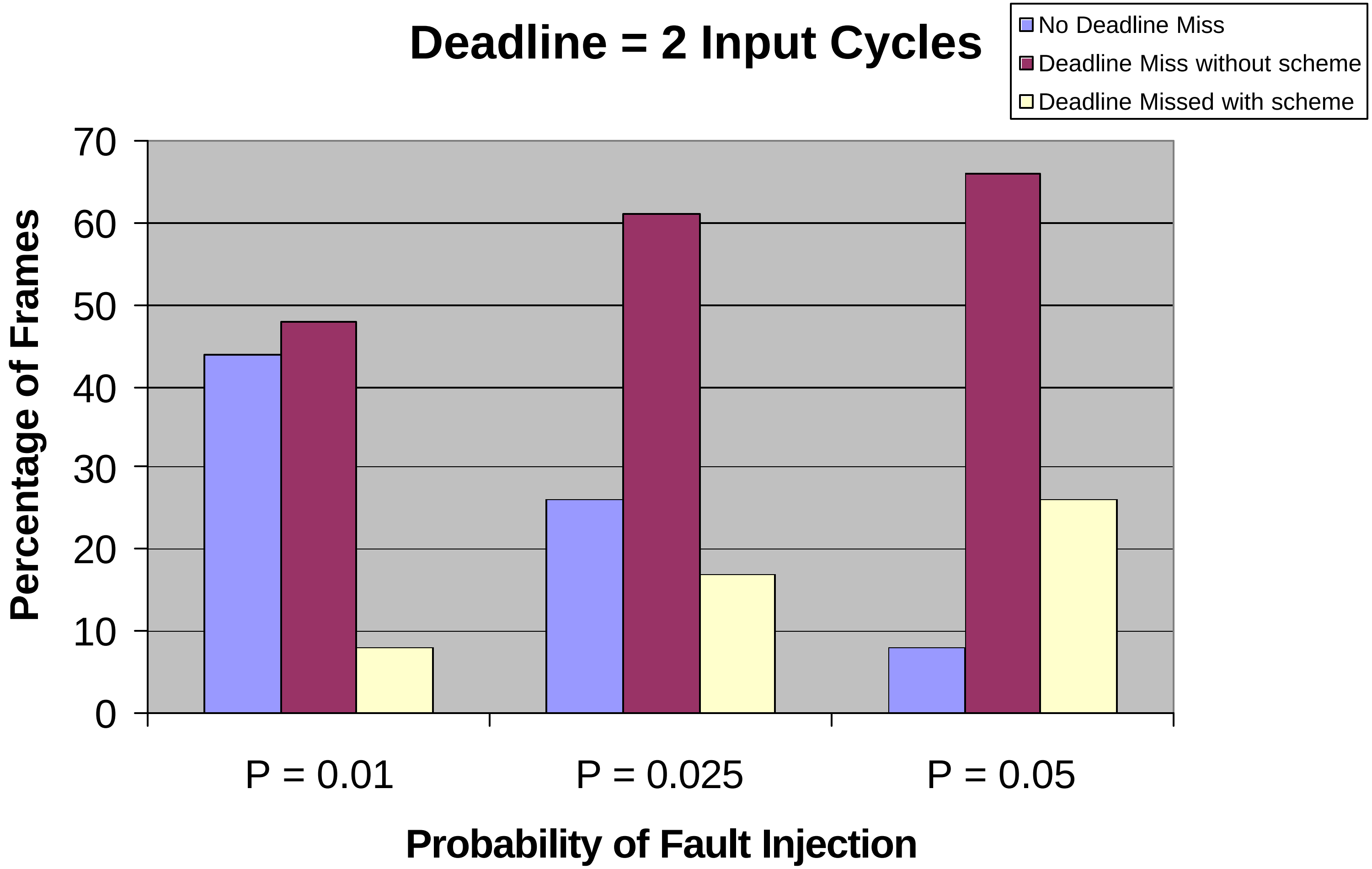
Integration Process:

- Receives all the processed frames
- Integrates them to form the composite OTIS FITS file
- Stores the file locally with proper filename to simulate the Down-linking to earth station

Deadline Miss Avoidance



Deadline Miss Avoidance



Conclusions

- By using the network processing potential to
  - Do a dynamic data analysis in constant time,
  - And then using the garnered run-time estimates,
  - It is possible to do intelligent scheduling of a Distributed Real Time System
- Substantial Reduction in the number of frames for potential deadline misses
- The original system design is oblivious – compatible and transparent to the scheme

Thank You