INSTITUTE FOR DEFENSE ANALYSES

Resilience Engineering Demonstrator System: Limited Bandwidth Text Analytics

(Presentation)

Jessica G. Swallow
Katherine I. Fisher
Nicholas J. Kaminski
Sarah L. Jones
Jeffrey A. Snyder

April 2021

IDA Document NS D-22638

Log: H 21-000137

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882

The Institute for Defense Analyses is a nonprofit corporation that operates three Federally Funded Research and Development Centers. Its mission is to answer the most challenging U.S. security and science policy questions with objective analysis, leveraging extraordinary scientific, technical, and analytic expertise.

About This Publication

This work was conducted by the Institute for Defense Analyses (IDA) under contract HQ0034-19-D-0001, Project AK-2-4793 "Resilience Engineering for DoD Software and Software-based Systems," for the Office of the Deputy Under Secretary of Defense for Research (DUSD/Research). The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

For More Information

Jeffrey A. Snyder, Project Leader
jsnyder@ida.org, (703) 578-2838

Leonard J. Buckley, Director, Science and Technology Division
lbuckley@ida.org, (703) 578-2800

Copyright Notice

Rigorous Analysis | Trusted Expertise | Service to the Nation

Executive Summary

Resilience engineering is a method that allows automated detection and correction of problems in an operational system in real time. These problems may arise as the environment changes or as the user's needs change. To demonstrate resilience engineering, IDA has built a proof-of-concept system. The demonstration system is a tool that collects multiple data feeds for an analyst that is looking at a specific subject area. There is limited bandwidth for processing, so the data feeds must be throttled to fit within the bandwidth limits. This briefing shows the demonstration system and how resilience engineering improves the performance of the system.

Contents

Resilience Engineering Demonstrator System: Limited Bandwidth Text Analytics (LBTA)

Jessica Swallow, Katherine Fisher,
Nicholas Kaminski, Sarah Jones, Jeffrey Snyder

24 March 2021

1. Background

Background

- Concept of resilience engineering (formerly known as AI Engineering) developed to allow automated detection and correction of problems in the operational system.
- Critical DoD missions rely on a complex set of interrelated systems:
  - A minor failure in any system can potentially lead to a mission failure
    - "For want of a nail …"
  - No clear one-to-one correspondence between system failures and mission failures.
- No routine way to design, deploy, monitor, and secure software-based systems.
- Artificial intelligence and machine learning are capabilities, not solutions to the problem.

This slide describes the resilience engineering background and the reasoning behind why the Department of Defense (DoD) needs to develop resilience engineering techniques to apply to DoD systems.

In this project we were tasked with building a system to demonstrate how resilience engineering would actually work. This demonstration system has to be complicated enough that resilience engineering would actually be useful, but not so complex that it would not be clear to the user how and why the resilience engineering is taking the actions it does.

Goal

- Prepare an unclassified, nontrivial (but not complex) system that demonstrates a resilience engineering proof of concept.

- Demonstration system is a tool that collects multiple data feeds for an analyst.
- Limited bandwidth prevents all data in all feeds from being used by analyst.
- Fixed, equal bandwidth allocation to each feed is suboptimal, especially as content of each feed changes or if interests of analyst changes.

The goal is to build a demonstration system as a resilience engineering proof of concept. It must be complicated enough to benefit from resilience engineering, yet still be simple enough that people can understand what the resilience engineering is doing and why.

We have chosen to look at the typical problem of a tool that collects multiple data feeds for an analyst that is looking at a specific subject area. There is limited bandwidth for processing, so the data feeds must be throttled to fit within the bandwidth limits. We do not concern ourselves with why the bandwidth limits exist. There are data streams that the analyst will find useful, and data streams that the analyst will not find useful. How the bandwidth is allocated will therefore have a significant impact on the overall utility of the system to the analyst.

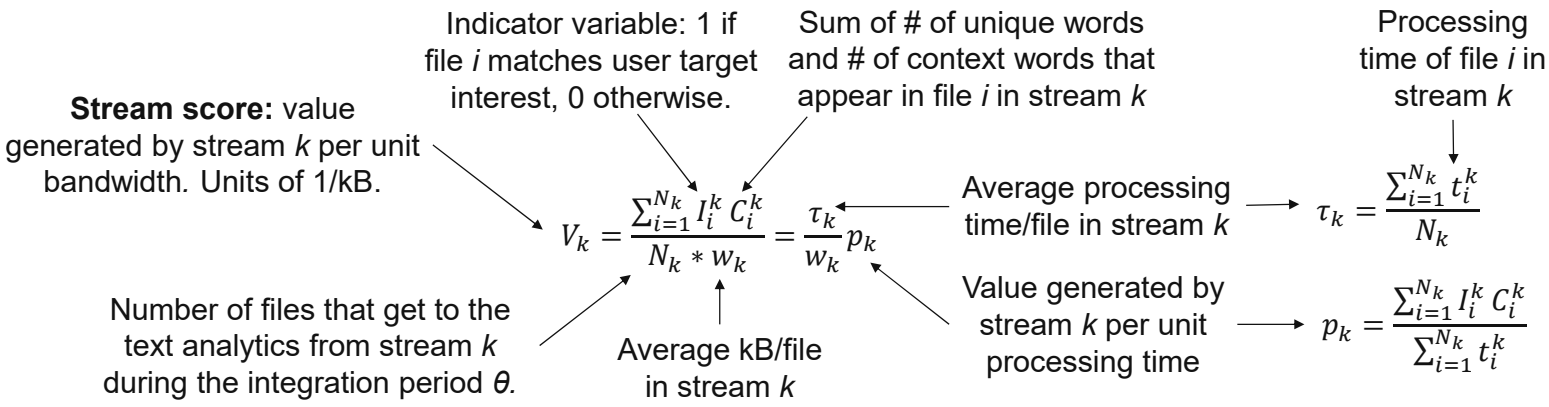
Text Analytics Concept

- Postulate a system with limited bandwidth where a user wants to collect emerging information about a topic:
  - Users choose between two topics: Cancer and natural disasters (or can enter their own).
  - Data sources: news headlines, research publications, social media posts, etc.
- Bandwidth limitations prevent system from processing full volume of each data source.
- Key metric is “value” of data stream for user’s chosen topic (see next slide):
  - Using simple text analytics (occurrence of keywords and relevant context) and processing time to calculate value.
- Orchestrator monitors data stream performance and optimizes total “value” of combined data streams, within bandwidth limit.
- Orchestrator can control:
  - Bandwidth allocation for each data source (based on calculated value).
  - When to reconfigure or reboot the system.
- Test harness allows data streams to be controlled separately and be “poisoned” during the demonstration.

For our demonstrator, we have postulated a system in which we want to conduct continuous data collection and processing from a number of data streams, but where we have only limited bandwidth to do so. Hence the system needs to be able to prioritize its available data sources based on their value to the user. In this example, the data sources are all text of some form, and we use a text-analytics processing approach to evaluate each data file and determine its value for the user. There are a few ways that value can be determined. For example, does it have the keywords the user is interested in? Does it have relevant context words surrounding those keywords? How long does it take to process each data set? The system has an orchestrator that is monitoring the performance of each data stream and determining when adjustments in bandwidth allocation are necessary, either by rebooting the system or reallocating bandwidth across the streams.

In this project we are focusing on the resilience engineering techniques, not the text analytics techniques. Therefore, we have chosen to use basic text analytics rather than developing a complex, nuanced text analysis.

Metrics



***Ψ* = System value** during integration time *θ*:

# of active data streams

Files/time of stream *k* sent to text analytics.

$$\Psi = \sum_{k=1}^{N} p_k N_k \tau_k = \sum_{k=1}^{N} V_k w_k N_k = \sum_{k=1}^{N} V_k w_k \dot{G}_k \Theta$$

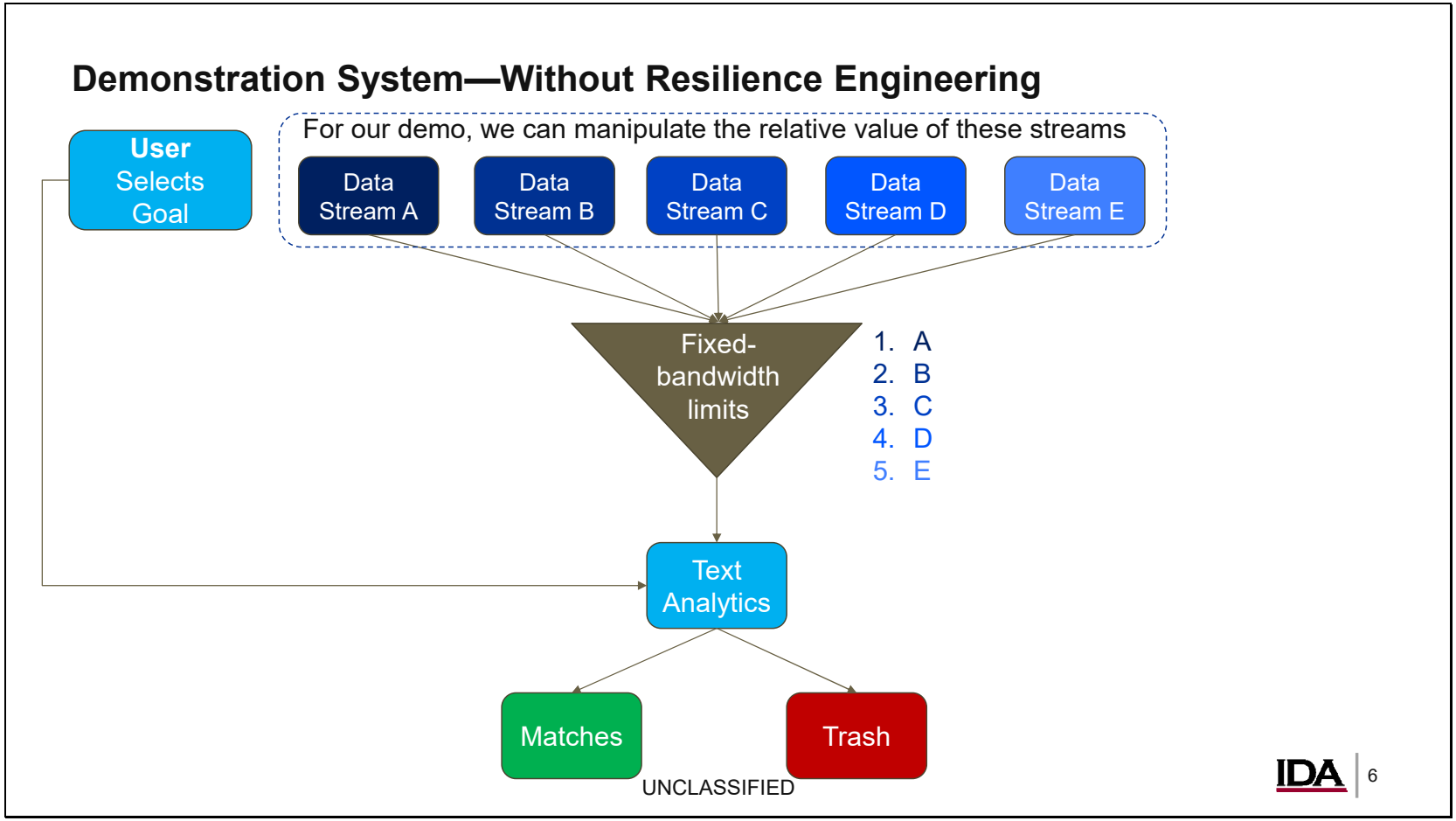
This chart summarizes the metrics we have chosen to use to evaluate stream and system performance. Stream performance is continuously evaluated over integration times, which are periods of data collection between points when the orchestrator chooses actions. At the end of each integration period, the orchestrator evaluates information about individual stream performance and overall system health (e.g., measures of data feed rates, confirmation that all portions of the system are functioning as expected) to decide on an action.

The score of stream k is given by V_k, which is calculated based on several pieces of information determined through the orchestrator's monitoring capabilities. Individual files that reach the text analytics for stream k are scored based on (1) whether any of the target keywords (as defined by the user goals) appear in the text and (2) the number of unique words in the file and the number of times key context words appear in the file. The number of unique words is used as a crude indicator of informational content. The scores of all files collected during the integration period are summed, and their sum divided by the total bandwidth that was occupied by stream k during the integration period. This penalizes data streams that have a high bandwidth per file but a low score per file.

Note that this method of assessing stream performance works out to value generated per unit bandwidth. Alternative metrics can be constructed, for example, value generated per unit processing time.

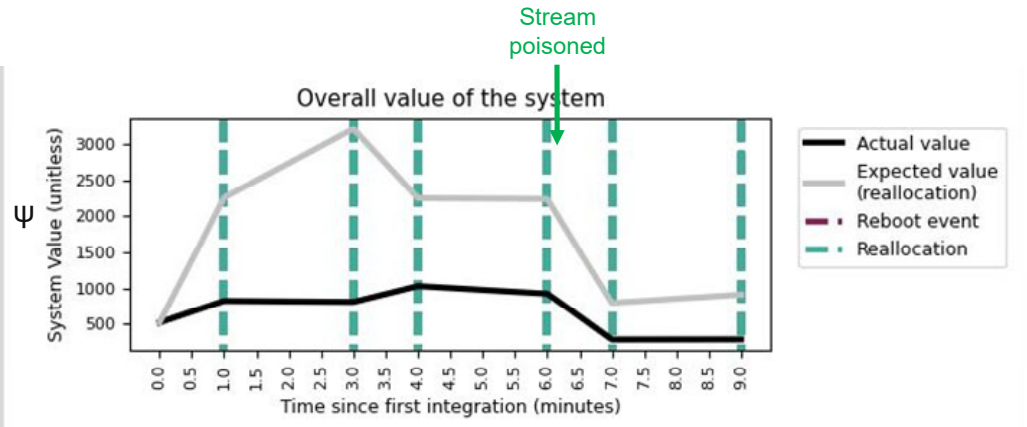
Overall system performance during an integration time is defined by the metric ψ, which sums the individual stream scores V_k weighted by the number of kilobytes that they used during the integration period.

2. Demonstration System



Without resilience engineering, the available data streams overfill the available bandwidth and have to be limited. Since the value of each data streams relative to the user's goal is unknown, the bandwidth limits are set to fixed, equal values. In some cases this works well and the total value of the combined streams is fairly high. In other cases this does not work well and the total value is suboptimal because low-value streams use too much bandwidth and do not leave enough bandwidth for high-value streams.

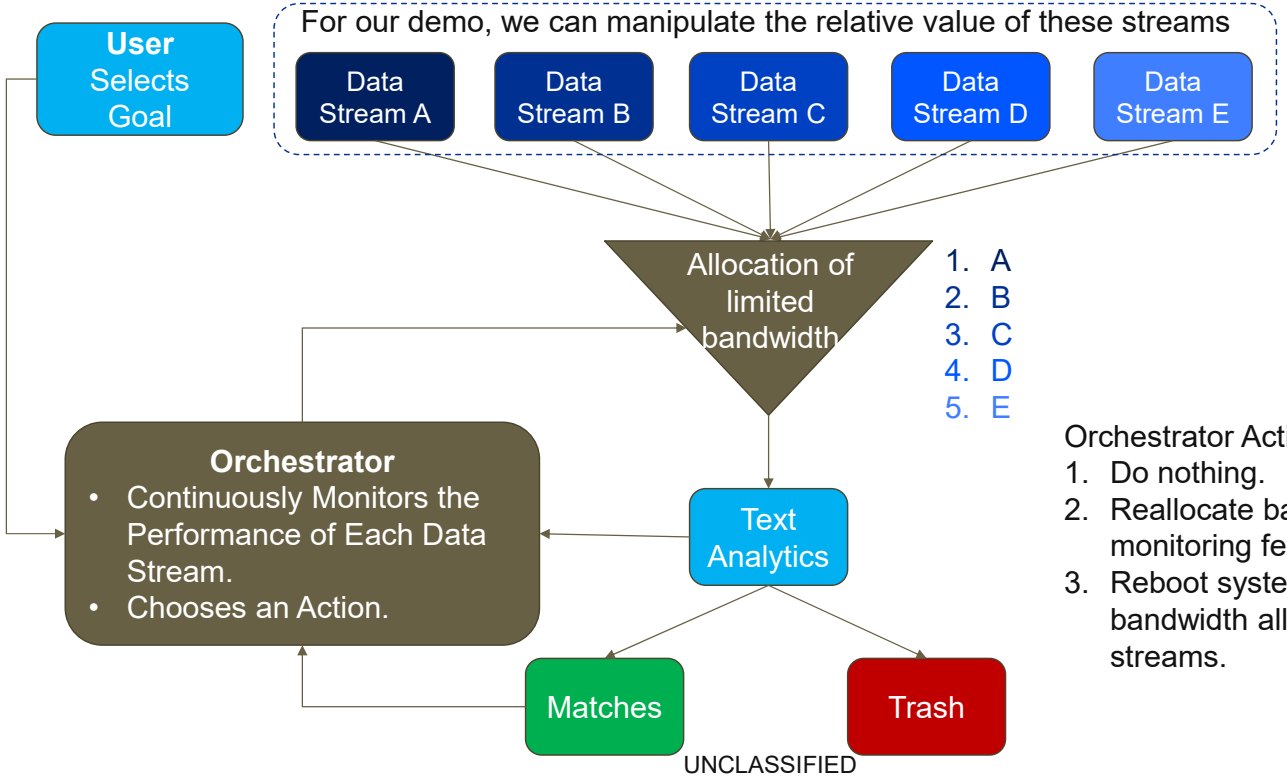
Example—System Performance without Resilience Engineering



This plot shows the overall performance of the system without the benefit of the orchestrator during a run in which the user poisoned one stream midway through the run. Note that the reallocations (dashed green lines) were not done because the orchestrator was disabled.

This chart shows an example of a run of the demonstrator where the orchestrator was disabled. The user poisoned one stream midway through the run. Plotted is the system performance as a function of time. The black line is the actual measured system performance ψ in the previous integration period, while the gray line indicates the orchestrator's estimated system performance if the proposed reallocation is accepted. The vertical green dashed lines indicate points where the orchestrator would have implemented a reallocate action had it been enabled. The actual system performance is significantly lower than the expected performance if bandwidth were optimally allocated.

Demonstration System—With Resilience Engineering



Orchestrator Actions:

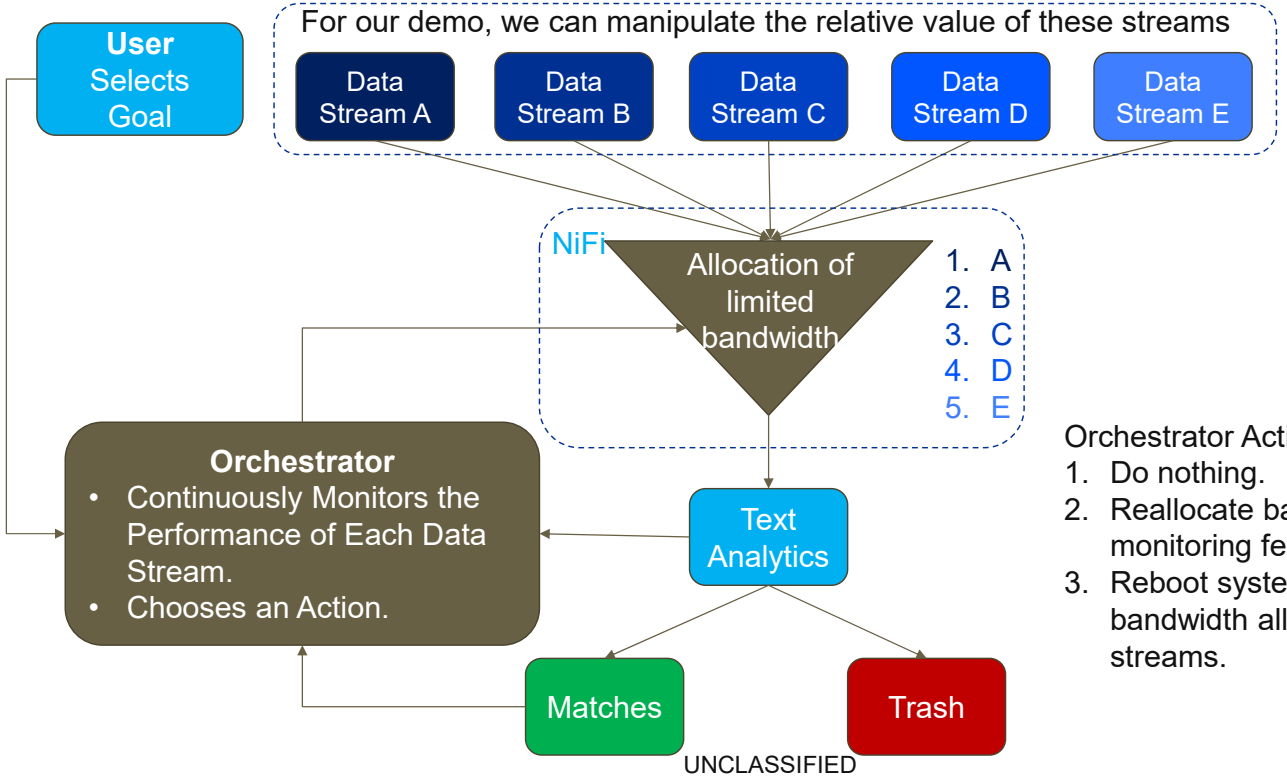
1. Do nothing.
2. Reallocate bandwidth based on monitoring feed.
3. Reboot system with equal bandwidth allocations for all streams.

IDA | 8

UNCLASSIFIED

Here is a general schematic of the system with resilience engineering. You can see that the user selects a goal through a user interface. The orchestrator then uses this goal to initially configure the system, in which multiple data sources are being funneled to our limited processing bandwidth. Successful files are stored, unsuccessful files are thrown out, and information about the value of each file is constantly reported to the orchestrator, which decides whether to take any action. The orchestrator has three actions at its disposal: it can do nothing, it can reallocate bandwidth across the data streams based on the monitoring information, or it can entirely reboot the system. Reallocation occurs when the orchestrator computes that a more optimized allocation will achieve a meaningful performance boost; rebooting may occur if some data streams appear blocked or the full system is detected to be non-functioning. A more advanced system could, for example, use alternative text analytics to improve the performance of the overall effort to collect information—but for our demo, we focus on the reallocate/reboot options.

Demonstration System—With Resilience Engineering



Data feed options:

- Lines from Shakespeare
- News Headlines (various sources)
- Tweets
- Academic journal abstracts

Orchestrator Actions:

1. Do nothing.
2. Reallocate bandwidth based on monitoring feed.
3. Reboot system with equal bandwidth allocations for all streams.

There are several options available for the data feed, including lines from Shakespeare, headlines from the news, tweets, and abstracts from academic journals. All the data are publicly available on the internet and do not contain PII. Through the test harness you can adjust the data rates of the streams, as well as bias their content toward one topic or another. This means that you can cause streams to become more or less valuable relative to the selected goal, which the orchestrator needs to be able to detect and adapt to. Data-feed handling and bandwidth allocation occur in NiFi.

NiFi is an open-source software suite that automates the flow of data between software systems. These flows can be controlled through a graphical user interface (GUI) (see slides 26–27) or an XML description of the interfaces.

Bandwidth Allocation

- All streams always granted at least a minimum bandwidth.
- Orchestrator proposes remaining bandwidth allocations according to the squared relative performance of each stream:

$$\frac{V_k^2}{\sum V_j^2}$$

  - If stream input rate is too low to fill its bandwidth allocation, extra gets passed to the next best stream.
  - Enforces some *fairness*. Resultant system value can be less than theoretical maximum because lower performing streams with nonzero value are still allocated some bandwidth above the minimum.
- Bandwidth allocations are only accepted if a substantial increase in system value is estimated.

Dynamic bandwidth allocation is done using an algorithm summarized here. All streams are granted a minimum bandwidth that permits them to always be "heard," even when they are very low scoring. This ensures that if a stream's performance changes, it will be detected, even if the stream was previously low performing. Bandwidth allocation then proceeds by apportioning remaining available bandwidth to streams based on their relative squared performance scores. This choice of weight ensures that the highest performing streams receive the most bandwidth, but also allows some fairness, in the sense that streams with moderate performance still receive more bandwidth than the minimum. If a stream does not have enough data flowing to fill its full bandwidth allocation, extra bandwidth is made available to the next best stream.

Based on the resulting proposed bandwidth allocations, the orchestrator then uses a model to estimate the expected performance ψ of the system if the bandwidth allocations are implemented. The orchestrator only implements a reallocation if a substantial improvement (>10% increase in ψ) is expected. Otherwise, the orchestrator does nothing.

Note that the orchestrator may also choose to reboot the system; slide 19 summarizes the conditions for a reboot. When a reboot occurs, equal bandwidth is allocated to all streams upon restart.

User Interacts with System Through a GUI

Test Harness

Change feed rate of data streams	Stop NiFi service	Pause processors within the NiFi template
Change or adjust content of data streams	Start new data streams	Stop existing data streams

User Actions

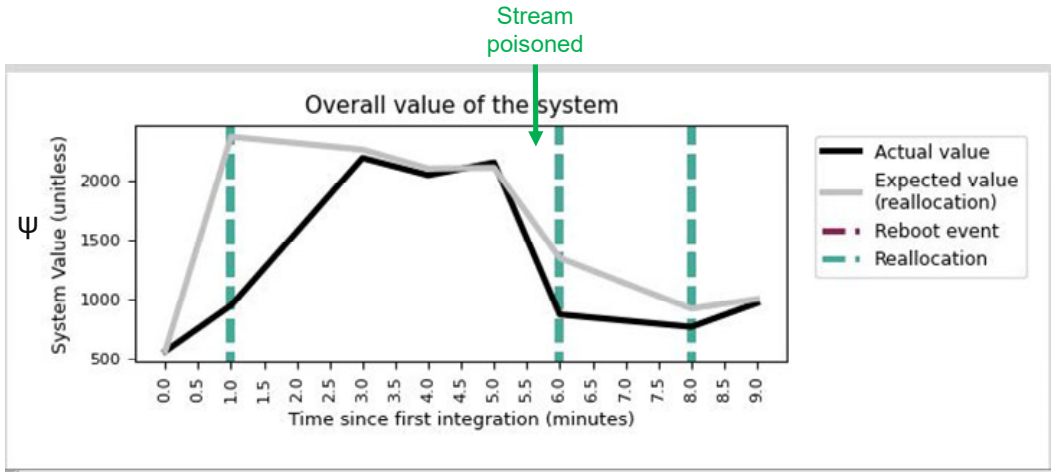
Change text analytics focus	Plot monitoring information	Stop Demo

- Red actions should trigger a reboot.
- Orange actions could trigger *either* a reboot *or* a reallocation.
- Yellow actions could trigger a reallocation.
- Green actions should not trigger any orchestrator actions.
- Determination of whether to reboot, reallocate, or do nothing depends on *monitoring information* available to the orchestrator.
- The orchestrator can *respond to* user actions *based on system monitoring*—but the user *does not* give the orchestrator instructions.
- Stopping the demo (gray) stops the orchestrator.

UNCLASSIFIED

This diagram represents all the actions that are available through the GUI, including actions designed to disrupt the activity of the NiFi system. The test harness actions are designed to mimic events that would be occurring in the environment, outside the control of the system user. In an operational context, these would be the events that a resilience engineering system would need to react to.

Example—System Performance with Resilience Engineering

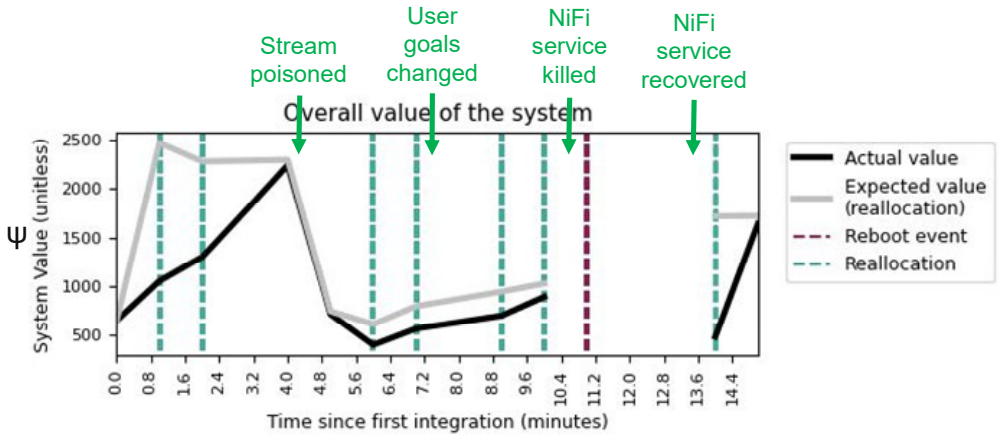


This plot shows the overall performance of the system with the orchestrator during a run in which the user poisoned one stream midway through the run. The actual system value (solid black line) is much higher than the performance without the orchestrator (shown on slide 7).

This chart shows an example of a run of the demonstrator where the orchestrator was enabled. The user poisoned one stream midway through the run. The system performance is plotted as a function of time. The black line is the actual measured system performance ψ in the previous integration period, and the gray line indicates the orchestrator's estimated system performance if the proposed reallocation is accepted. The vertical green dashed lines indicate points where the orchestrator implemented a reallocate action. After reallocation, the actual system performance approaches the performance expected if the bandwidth were optimally allocated. As shown in the plot, reallocation occurs only when a substantial gain in ψ is anticipated.

This example can be compared directly with the orchestrator disabled case shown on slide 7.

Example—System Performance with Resilience Engineering

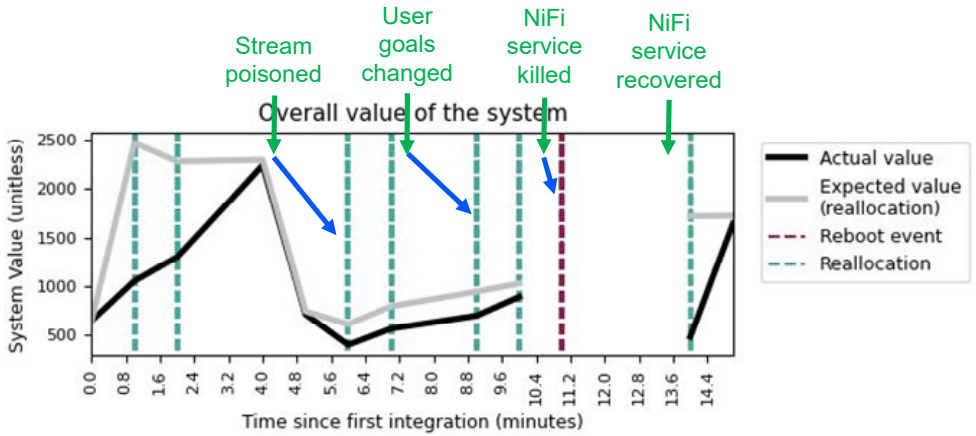


This plot shows the overall performance of the system during a run in which the user took several actions using the test harness: poisoning one stream, changing the goals for the text analytics, and killing the NiFi service.

This chart shows an example of a run of the demonstrator in which several actions are taken, including poisoning a stream, changing the user's goals, and killing the NiFi service in the orchestrator. System performance is plotted as a function of time. The black line is the actual measured system performance ψ in the previous integration period, and the gray line indicates the orchestrator's estimated system performance if reallocation is accepted. As shown in the plot, reallocation occurs only when a substantial gain in ψ is anticipated.

The vertical green dashed lines indicate points where the orchestrator implemented a reallocate action. The vertical dashed maroon line indicates a point when the orchestrator implemented a reboot action.

Example—System Performance with Resilience Engineering

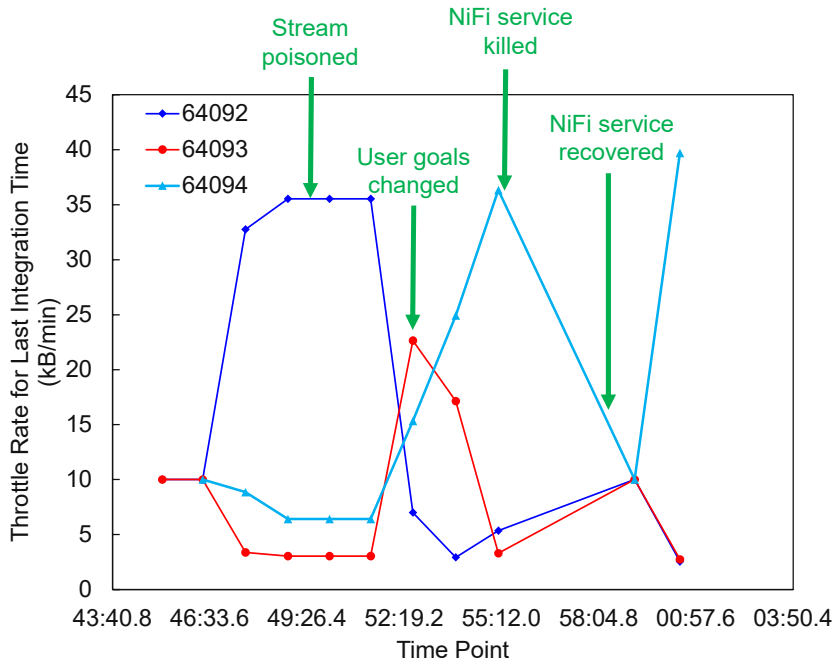


Blue arrows show the orchestrator's response to each external event. Notice how system value eventually improves after initial drops in performance due to poisoning or loss of NiFi service.

This chart highlights the orchestrator's actions in response to different events from the test harness. Shortly after a stream was poisoned, system performance significantly degraded, but this degradation was detected and the orchestrator reallocated bandwidth, slightly improving performance. After the user changed the goals of the text analytics, the orchestrator reallocated a few times, improving system performance. After the NiFi service was killed, the orchestrator rebooted the system, and performance was recovered after the first reallocation following that reboot.

This is a screenshot from the GUI—note that data are plotted with *x*-axis values rounded to the nearest minute.

Same Example—Bandwidth Allocations



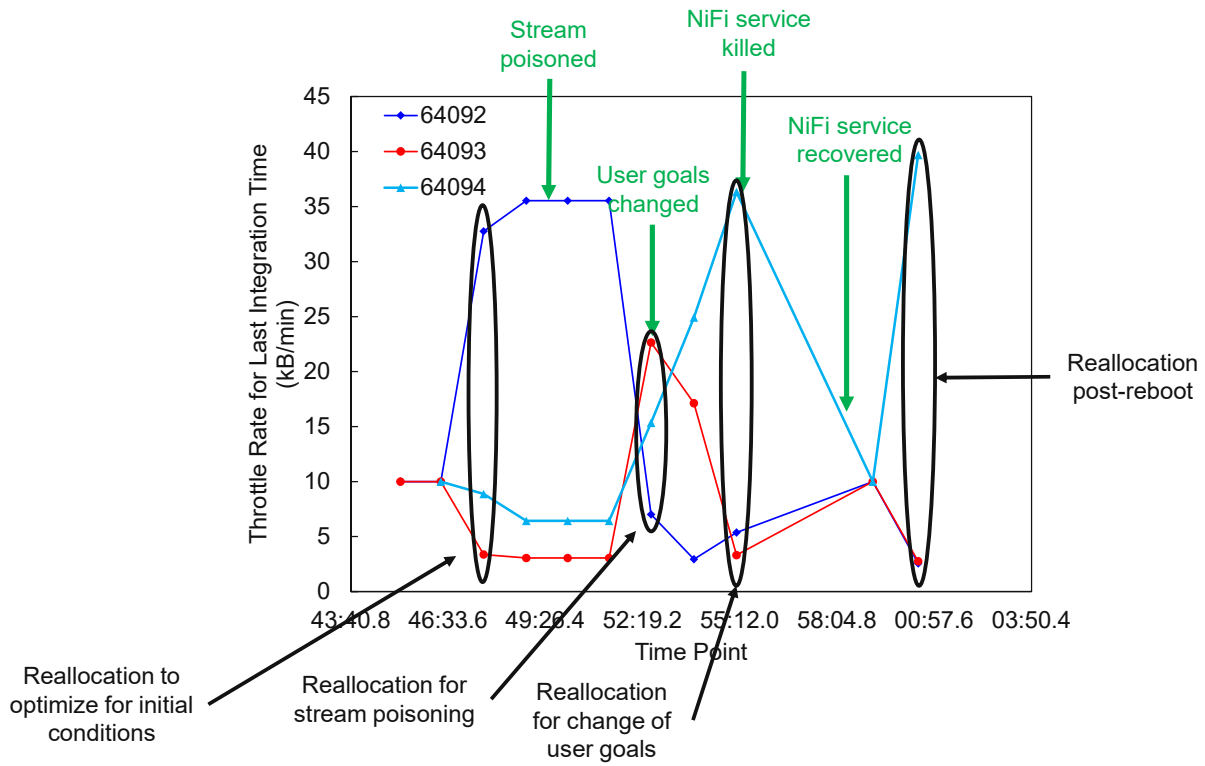
This chart reports the bandwidth allocations for the streams (designated by port numbers 64092–64094) as a function of time.

Note that *this chart lags the real-time status of the system*—the data points represent the bandwidth allocation during the previous integration period.

It typically takes about two integration times from the time of an external stimulus for the resulting system adaptation to show on this chart. This is the time for the system to detect the change, take an action, and then plot that action at the end of the new integration time.

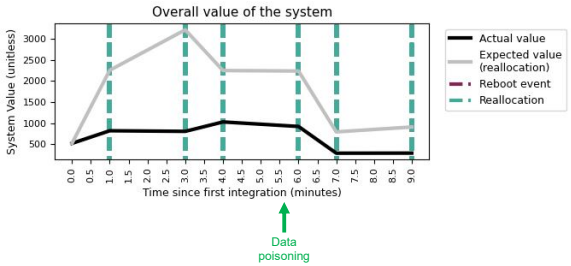
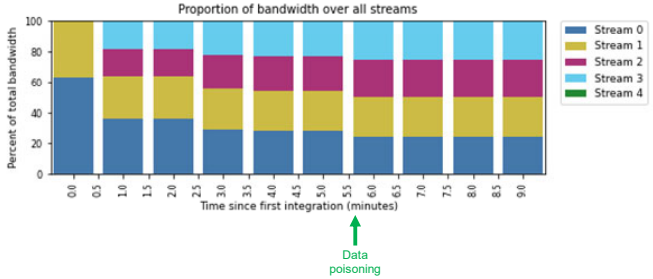
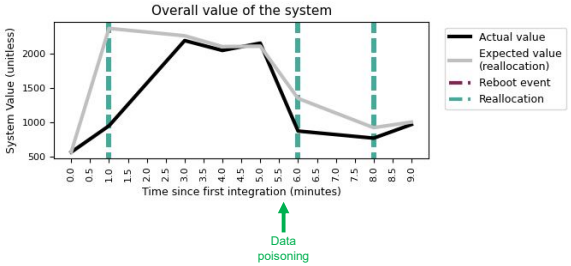
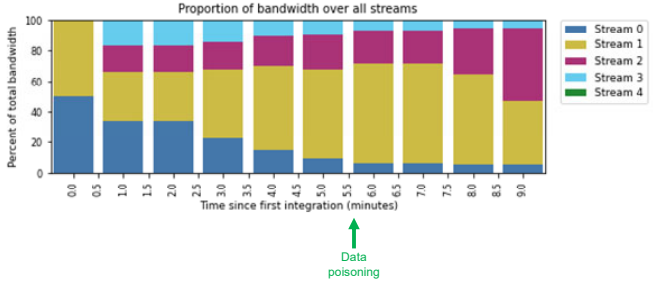
This chart plots throttle rates for each data stream as a function of time for the same demonstrator run as shown on the previous two slides. The throttle rate is the permitted bandwidth per minute for each stream. These data are the rates for the previous integration time, so they lag real-time events by about a minute. As noted on the chart, it typically takes about two integration times from the time of an external stimulus for system adaptation to be apparent on this chart. This is the time required for the system to detect the change in performance, take a corrective action, and then collect another integration period's worth of data to plot the results.

Same Example—Bandwidth Allocations



This chart highlights the detail of certain reallocation events that occurred in response to various external events during the demo shown on the prior slides. The initial system performance was optimized by giving stream 64092 the most bandwidth; however, after that stream was poisoned, streams 64093 and 64094 were given more bandwidth. Then, when the user's goals were changed, stream 64094 was given the most bandwidth. After the system reboot occurred, all streams were initialized with equal bandwidth, but the orchestrator quickly detected that stream 64094 was the best performing stream and gave that stream the largest bandwidth allocation again.

Comparison: System with and without Resilience Engineering

	System Value (ψ)	Bandwidth Used
Orchestrator Disabled		
Orchestrator Enabled		

This chart compares the overall system value (ψ) (left column) and bandwidth used by each stream (right column) between nearly identical cases with the orchestrator disabled (top row) and enabled (bottom row). The system value is much higher when the orchestrator is allowed to increase the bandwidth allocated to the more valuable streams at the expense of the bandwidth allocated to less valuable streams. This is also true after data poisoning (around 5.5 minutes). Note that the system automatically detects and corrects for the data poisoning with no need for human intervention.

Note that the first two entries on the bandwidth used plots are incomplete because it took over 30 seconds to get all the data streams flowing. The target topic was “cancer” with context words “treatment” and “therapy.” Poisoning was implemented 4 minutes and 40 seconds after initialization of stream 3 and consisted of changing the natural hazard bias of stream 1 to 91%.

Initial stream specifications:

0: Shakespeare 6 Hz

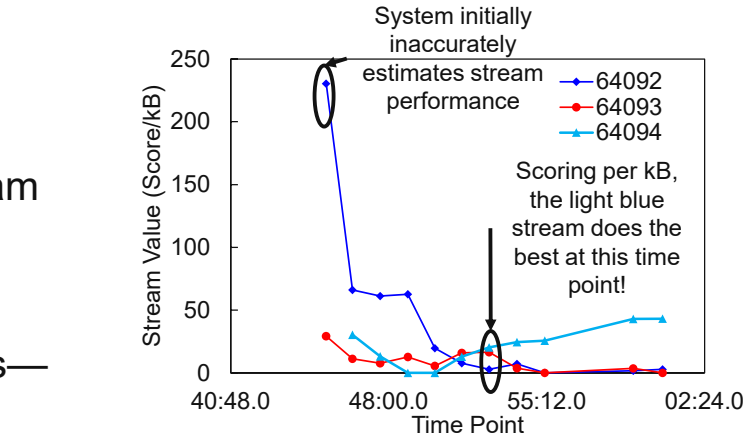
1: Twitter, 10% natural hazard bias, 4 Hz

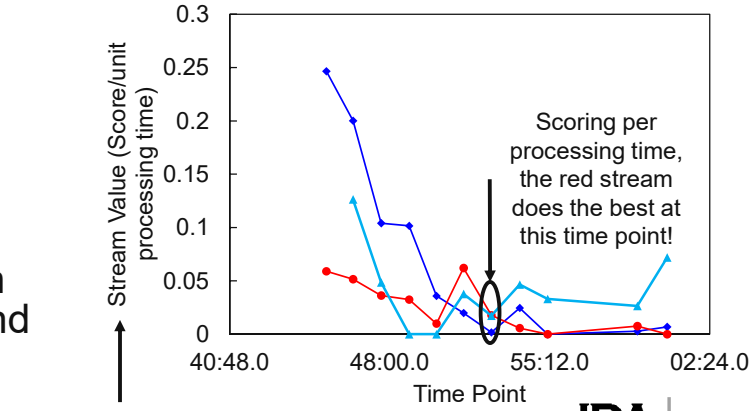
2: Twitter, 60% natural hazard bias, 4 Hz

3: NewsCat, 5 Hz

Role of Metric Selection

- Two different ways of measuring stream score:
  - Normalized by stream *processing time.*
  - Normalized by stream *bandwidth* (*kB*).
- *Usually* doesn't affect stream rankings—but it can!
- Often leads to different *relative* performance.
- Choice of metric will influence the orchestrator's suggested reallocation scheme!
- Also, note initial inaccuracy of stream scores: the system hasn't had enough time to collect performance metrics and has to make some assumptions!





This metric is usually more volatile

This slide highlights the effect of the choice of stream performance metric on the orchestrator's choices. As noted on slide 5, value/bandwidth was the metric used by the orchestrator to judge stream performance. Alternative metrics exist, however, such as value/processing time. These metrics *usually* result in the same stream rank, but they often show different relative performance (which would affect exact bandwidth-allocation weights), and they occasionally result in different stream ranks, as shown here near minute 53. Scoring per processing time is also typically more volatile than scoring per bandwidth, as shown in the graphs on this slide. Thus, the choice of metric influences the orchestrator's proposed reallocations and its ability to maximize system performance. Note also that scoring of the streams is initially quite inaccurate because the orchestrator hasn't yet had enough time to collect performance information about the individual streams and is therefore forced to make some assumptions about their performance.

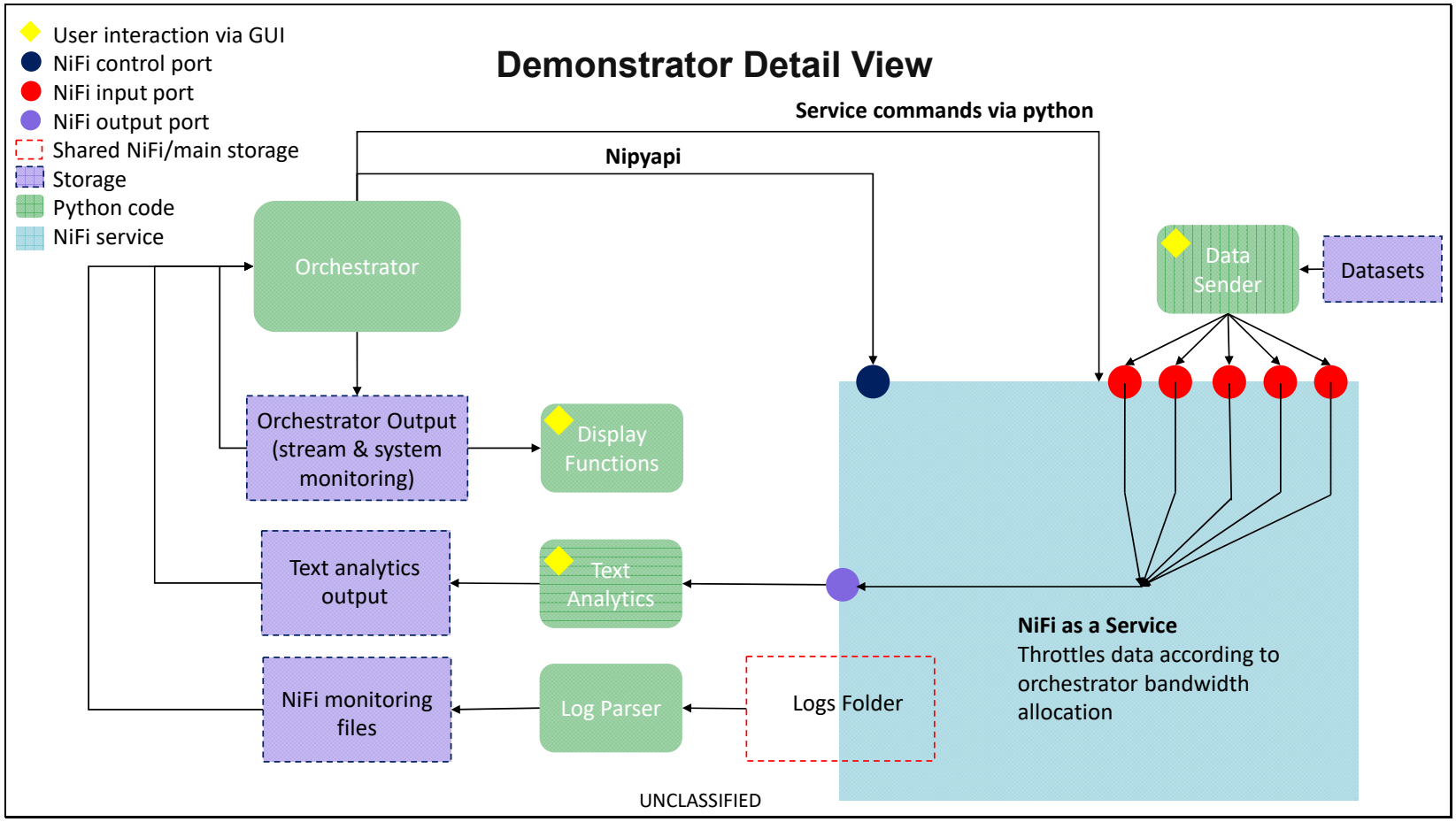
Conclusion

- We have created an unclassified, nontrivial (but not complex) demonstration system that is capable of demonstrating the ideas of resilience engineering.
  - The Low-Bandwidth Text Analytics (LBTA) system incorporates an orchestrator that optimizes the bandwidth allocations of different data feeds based on the analyst's current interest and the data feeds' recent value to that interest.
- The resilience engineering (RE) concepts demonstrated in LBTA can be applied to other systems, such as:
  - Communications networks.
  - Sensor networks.
  - Programming code.

Appendix A.
Additional Detail on the Limited Bandwidth Text Analytics (LBTA) Demonstrator System Construction and Operation

Appendix: Additional Detail on the LBTA Demonstrator System Construction and Operation

The remaining charts provided additional detail about the overall architecture and operation of the LBTA demonstrator.

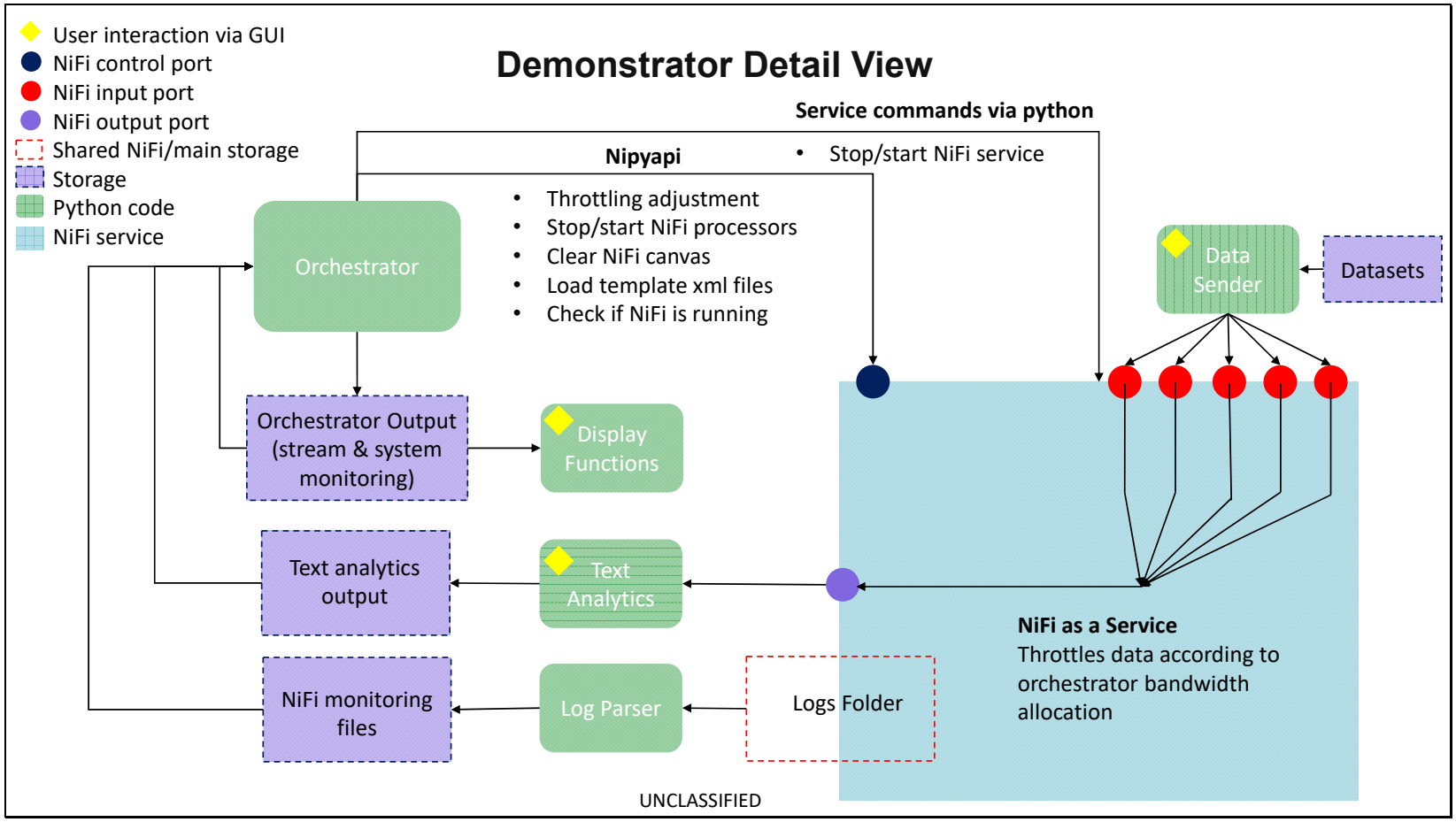

User interaction via GUI
NiFi control port
NiFi input port
NiFi output port
Shared NiFi/main storage
Storage
Python code
NiFi service
Demonstrator Detail View
Service commands via python
Nipyapi
Orchestrator
Data Sender
Datasets
Orchestrator Output (stream & system monitoring)
Display Functions
Text analytics output
Text Analytics
NiFi monitoring files
Log Parser
Logs Folder
NiFi as a Service
Throttles data according to orchestrator bandwidth allocation
UNCLASSIFIED

This schematic is a wiring diagram representing the full system. The full system runs inside a docker container that can be hosted on a Linux machine or Amazon Web Services node. The goal is to show how the orchestrator and other python code components interact with file storage and the NiFi service.

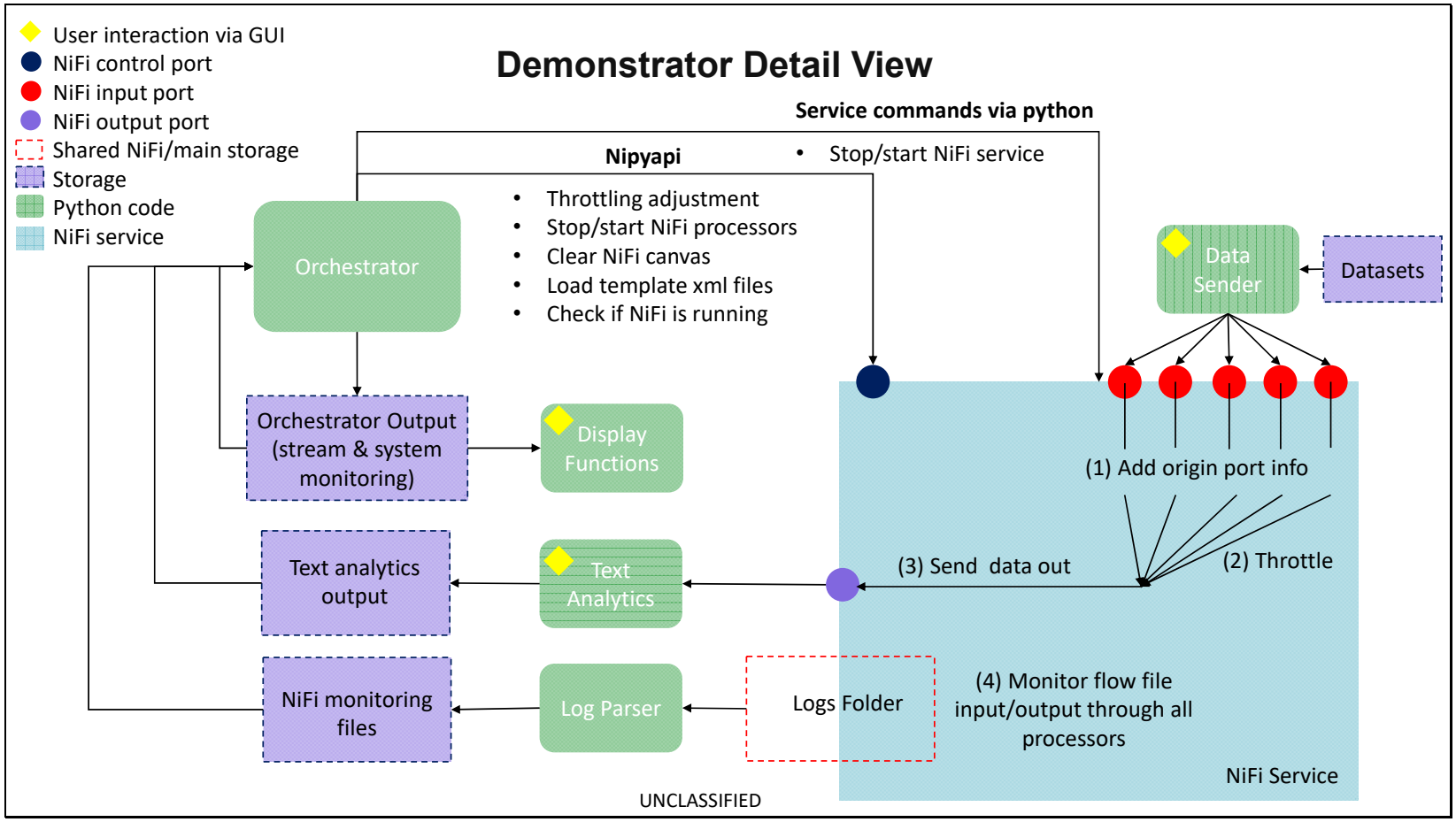
Arrows represent the direction of information flow, including generation of new files (arrows that end in the purple data storage rectangles), reading of files (arrows that begin in the purple data storage rectangles), movement of files (arrows moving from data sender through NiFi to text analytics output), and sending of commands (Nipyapi and NiFi as a service).

This diagram doesn't necessarily represent user NiFi-disruption functions, which also act via the NiFi control port or via service commands.

Interaction points available from the GUI are represented with yellow diamonds.



Here, we have added some information about the types of interactions the orchestrator has with NiFi.



Here, we show a little more detail about what is happening inside NiFi.

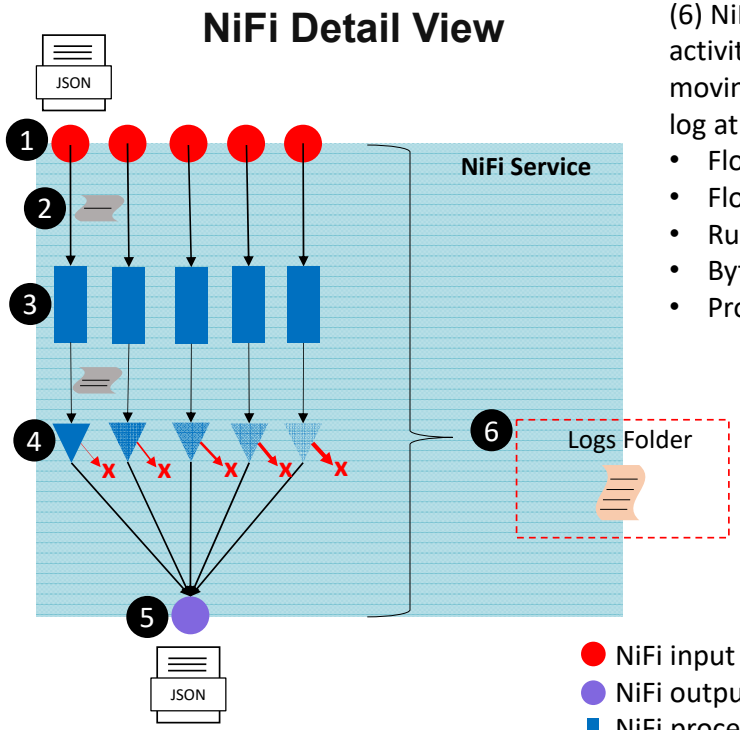
(1) JSON received via TCP

(2) Flow file created
- Time received
- Size received
- Port received from
- Content

(3) Flow file updated
- Port information added to content

(4) Throttling controls bytes/min for each data stream
(x) Flow files that wait too long are discarded

(5) JSON sent out via TCP



(6) NiFi monitors processor activity according to a 5-minute moving average—records in a log at regular intervals:
- Flow files/bytes in
- Flow files/bytes out
- Run status
- Bytes read/written
- Processing time/flow file

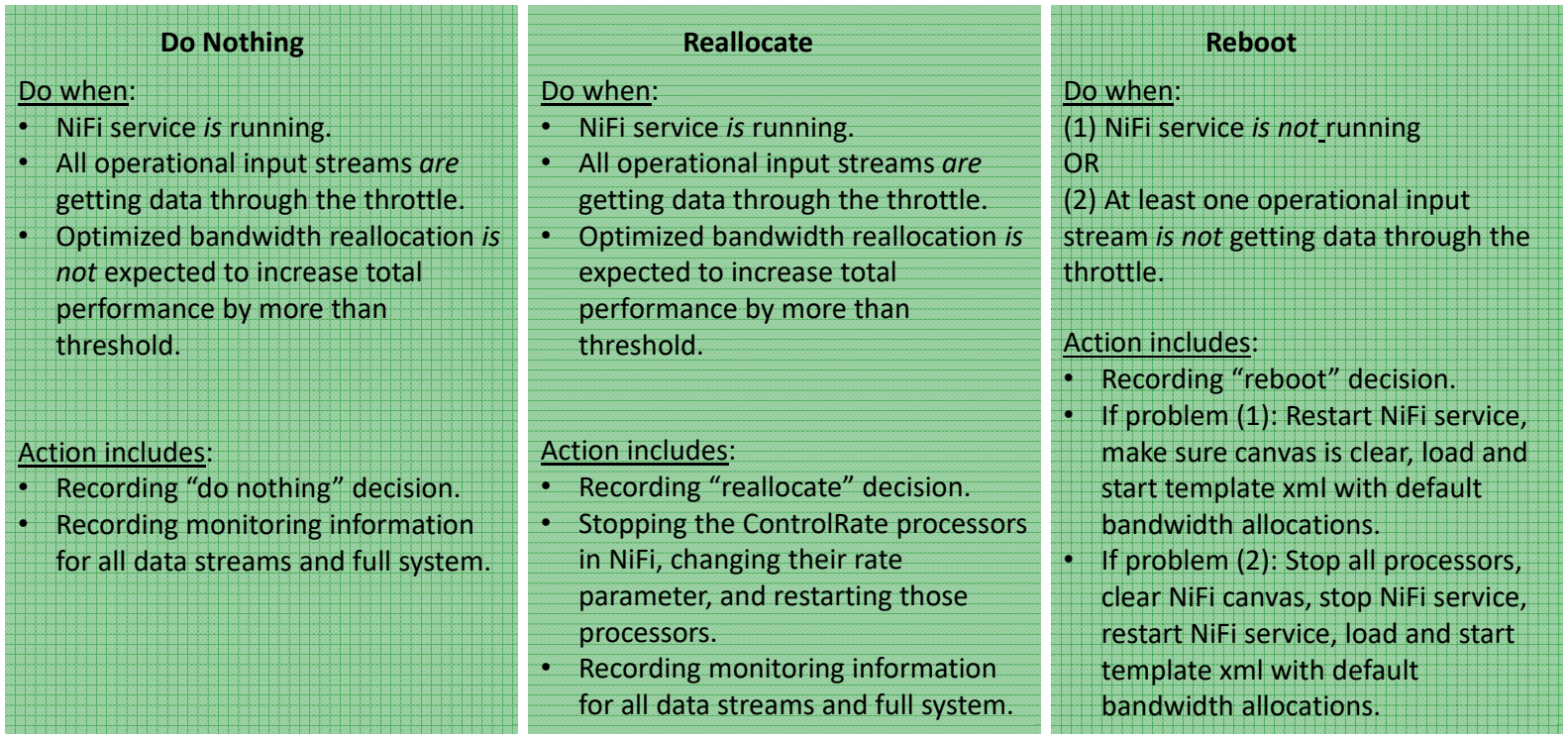
- NiFi input TCP port
- NiFi output TCP port
- NiFi processors
- NiFi ControlRate (throttle) processor

- Flow file
- JSON file
- Log file
- x Discard

UNCLASSIFIED

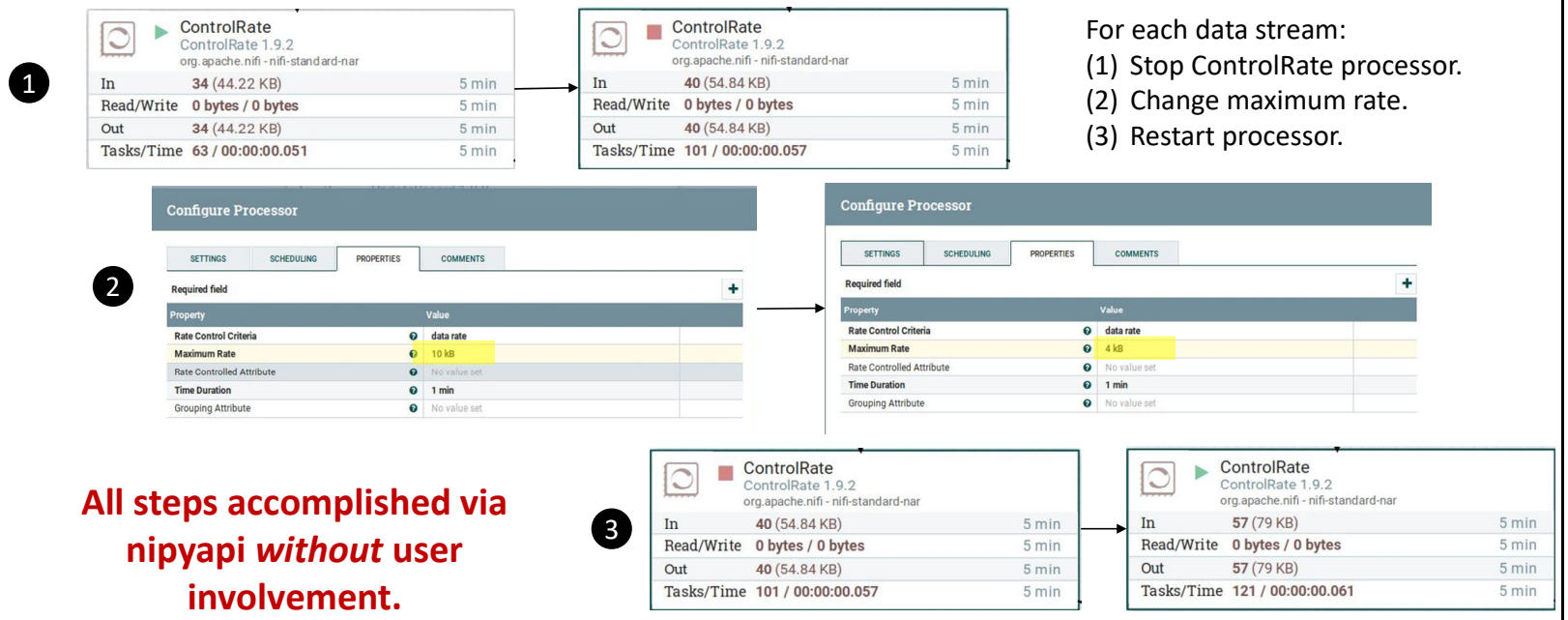
This schematic is a zoomed-in view of what happens inside of the NiFi service. Up to five data streams may be operational at any given time, but they will be throttled differently, depending on the performance of those streams detected by the orchestrator. In the diagram, darker shading means that more flow files are being let through for that data stream. Flow files that wait too long are discarded from the flow, as indicated by the red ×. Streams that experience more throttling (smaller proportion of flow files let through) will also have a higher proportion of files discarded (indicated by the thickness of the red arrows).

Orchestrator Actions and Decision Algorithm



This chart summarizes the orchestrator's available actions and the conditions that must be met to choose each action. These conditions are checked during each integration time using a decision algorithm.

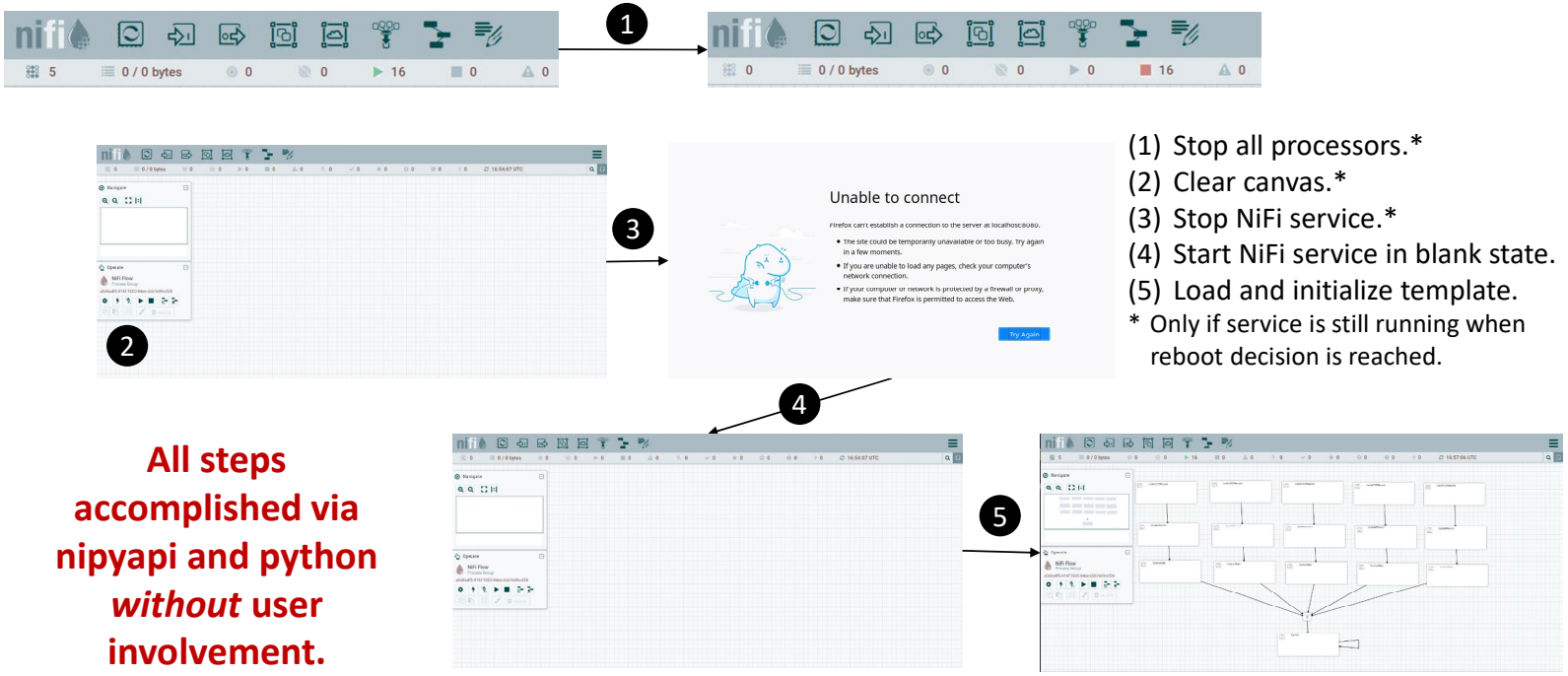
What is happening in NiFi during reallocation?



UNCLASSIFIED

This chart gives a more detailed view of what occurs in the NiFi service during a reallocation event.

What is happening in NiFi during a reboot?



(1) Stop all processors.*
(2) Clear canvas.*
(3) Stop NiFi service.*
(4) Start NiFi service in blank state.
(5) Load and initialize template.
* Only if service is still running when reboot decision is reached.

All steps accomplished via nipyapi and python *without* user involvement.

UNCLASSIFIED

This chart gives a more detailed view of what happens inside of NiFi during a reboot decision.

Orchestrator: Recorded Information After Each Decision Cycle

System Level

- Timestamps of decision-making
- **Reboot/Reallocate/Do Nothing** decision
- Allowed maximum bandwidth
- Actual bandwidth used
- System value generated during prior integration time
- **Expected value in next integration time if reallocation occurs**

For Each Data Stream

- Timestamps
- Stream score metrics (with and without processing time or kB normalization)
- Average text analytics processing time/file
- Feed rate in (files and bytes)
  - Two methods of estimating byte rate—the value used for orchestrator decision is also recorded.
- **Post-throttle feed rate** (**files** and bytes)
  - Two estimators for file rate based on different monitoring methods—both are recorded.
- **Proposed reallocation bandwidth limit**

When a REBOOT decision is reached, only RED items are recorded.
BLUE items are determined from NiFi monitoring logs.
PURPLE items are determined from text analytics results.
Bold items are calculated from **both** information sources.

UNCLASSIFIED

This chart summarizes information that the orchestrator tracks and records after each decision cycle.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED *(From–To)*
April 2021	FINAL	

4. TITLE AND SUBTITLE

Resilience Engineering Demonstrator System:
Limited Bandwidth Text Analytics (Presentation)

5a. CONTRACT NUMBER
HQ0034-19-D-0001

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Swallow, Jessica G.
Fisher, Katherine I. ,
Kaminski, Nicholas J.
Jones, Sarah L.
Snyder, Jeffrey A.

5d. PROJECT NUMBER
AK-2-4793

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Institute for Defense Analyses
4850 Mark Center Drive
Alexandria, VA 22311-1882

8. PERFORMING ORGANIZATION REPORT NUMBER

IDA Document NS D-22638

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)
USD(R&E)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited (25 May 2021).

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Resilience engineering is a method that allows automated detection and correction of problems in an operational system in real time. These problems may arise as the environment changes or as the user's needs change. To demonstrate resilience engineering, IDA has built a proof-of-concept system. The demonstration system is a tool that collects multiple data feeds for an analyst that is looking at a specific subject area. There is limited bandwidth for processing, so the data feeds must be throttled to fit within the bandwidth limits. This briefing shows the demonstration system and how resilience engineering improves the performance of the system.

15. SUBJECT TERMS

Limited Bandwidth; Natural Language Processing; Resilience Engineering

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Bonneau, Robert
Uncl.	Uncl.	Uncl.	SAR	54	19b. TELEPHONE NUMBER *(include area code)* (571) 372-6724