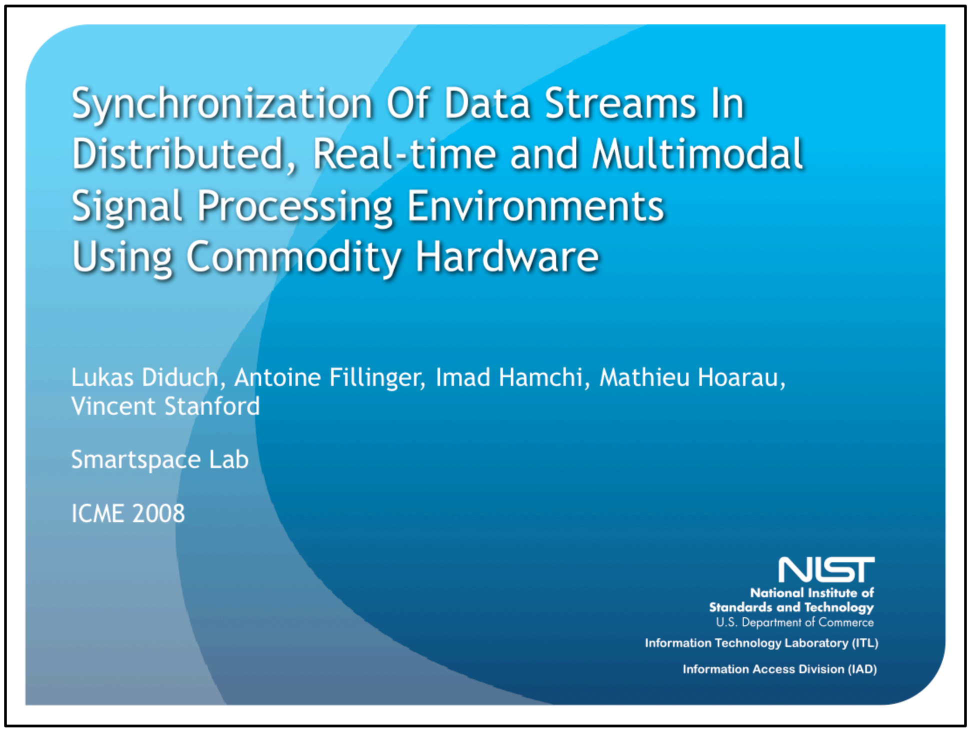


Synchronization Of Data Streams In Distributed, Real-time and Multimodal Signal Processing Environments Using Commodity Hardware

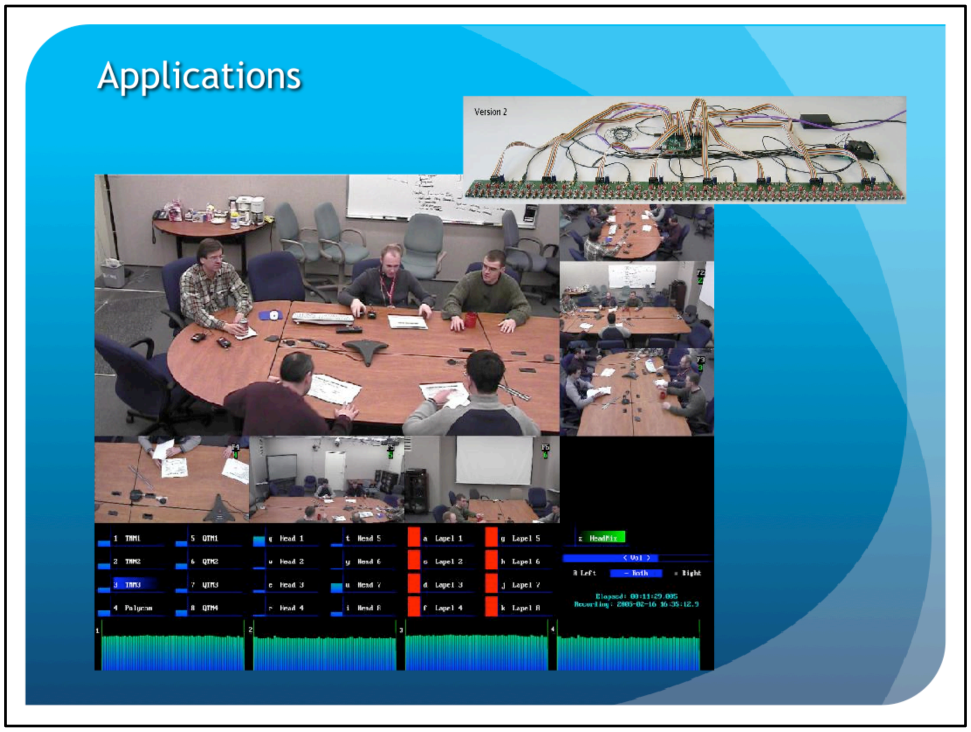
Lukas Diduch, Antoine Fillinger, Imad Hamchi, Mathieu Hoarau, Vincent Stanford

Smartspace Lab

ICME 2008

National Institute of Standards and Technology
U.S. Department of Commerce
Information Technology Laboratory (ITL)
Information Access Division (IAD)

ICME 08 presentation, session TM-AM1-L5.4 (June 26, 2008)



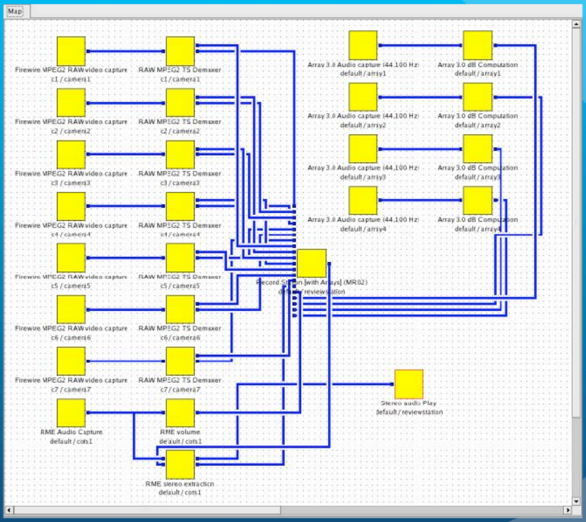
Applications

The big image is a screenshot of the ‘NIST meeting room Dist. Data acquisition task’ producing large multi modal corpora (10 TByte) used by various intl. research communities in context sensitive research (CHIL, CLEAR, VASE, AMI).
It represents a good example of a heavy real-time computational task (capturing about 200 GByte / h) which is distributed due to performance reasons, and more important due to availability of distributed sensors. It consists of over 280 microphones (4 arrays a 64 microphones + 24 direct mics) and 7 HD cameras.

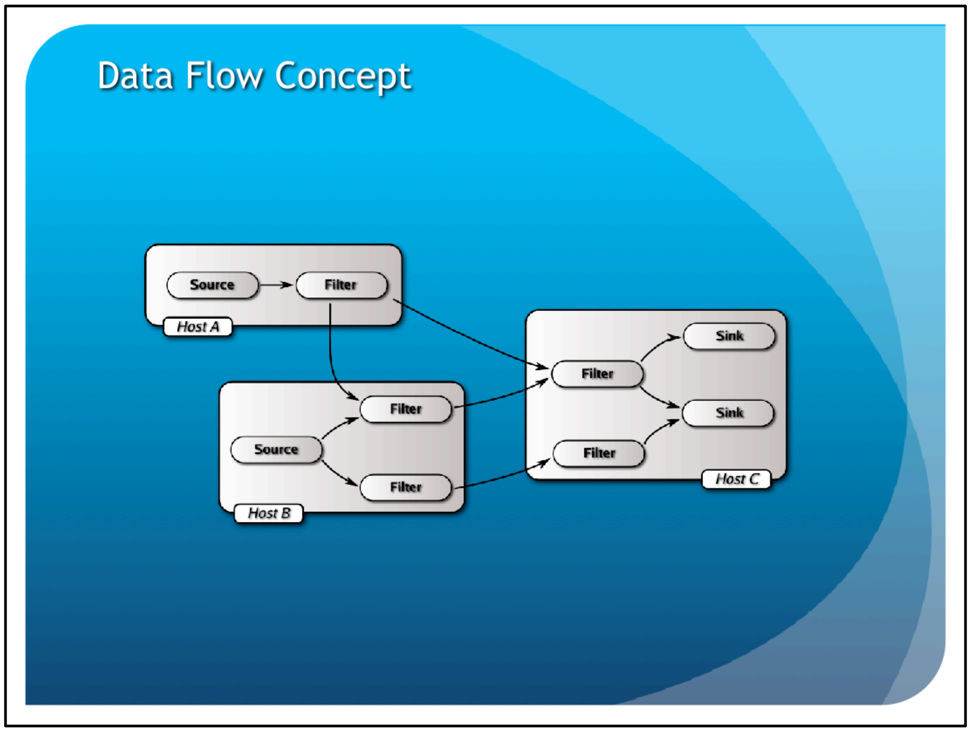
The smaller upper image is a photo of our in house developed 64 channel 24 bit 22-44 kHz MK-III microphone arrays, a Ethernet device.

This review station does not involve synchronization. It can not be guaranteed that one image of the HD views corresponds to the exact frame of the rest. Same applies to the microphone output data. This is not critical however, because the data is only recorded and viewed in this example. This setup is also working quick enough to make the impression of a synchronized display. Once processing in real-time is involved, synchronization becomes a crucial part of signal processing.

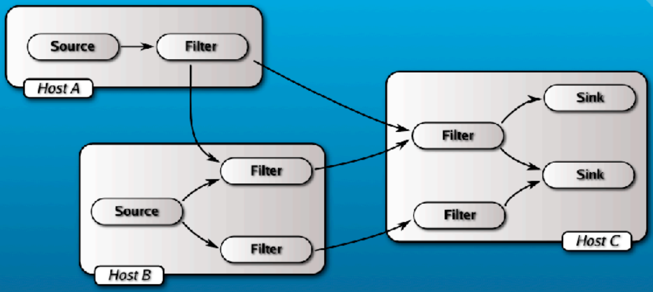
Applications


Map
Firewire MPEG2 RAW video capture c1 / camera1
RAW MPEG2 TS Demuxer c1 / camera1
Firewire MPEG2 RAW video capture c2 / camera2
RAW MPEG2 TS Demuxer c2 / camera2
Firewire MPEG2 RAW video capture c3 / camera3
RAW MPEG2 TS Demuxer c3 / camera3
Firewire MPEG2 RAW video capture c4 / camera4
RAW MPEG2 TS Demuxer c4 / camera4
Firewire MPEG2 RAW video capture c5 / camera5
RAW MPEG2 TS Demuxer c5 / camera5
Firewire MPEG2 RAW video capture c6 / camera6
RAW MPEG2 TS Demuxer c6 / camera6
Firewire MPEG2 RAW video capture c7 / camera7
RAW MPEG2 TS Demuxer c7 / camera7
RME Audio Capture default / cots1
RME volume default / cots1
RME stereo extraction default / cots1
Array 3.0 Audio capture (44,100 Hz) default / array1
Array 3.0 dB Computation default / array1
Array 3.0 Audio capture (44,100 Hz) default / array2
Array 3.0 dB Computation default / array2
Array 3.0 Audio capture (44,100 Hz) default / array3
Array 3.0 dB Computation default / array3
Array 3.0 Audio capture (44,100 Hz) default / array4
Array 3.0 dB Computation default / array4
Record Station [with Arrays] (MR02) default / reviewstation
Stereo audio Play default / reviewstation

This is a dataflow view of this task. You can see all sensor capturing nodes as well as the review station in the middle which fuses the data.



Data Flow Concept



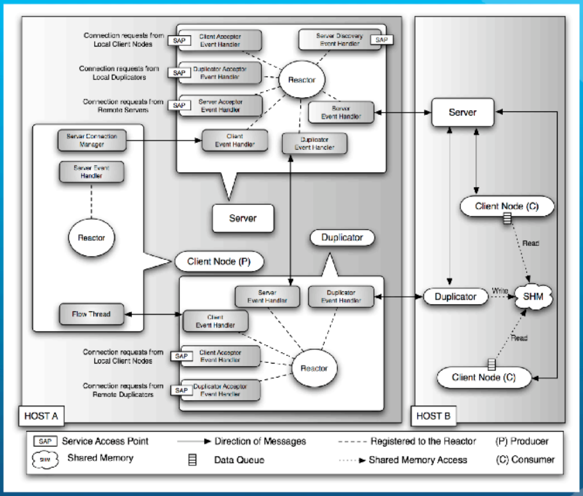
To process and capture data in a distributed environment we usually want to apply the concept of a DATAFLOW.

There are mainly three kinds of processing entities:

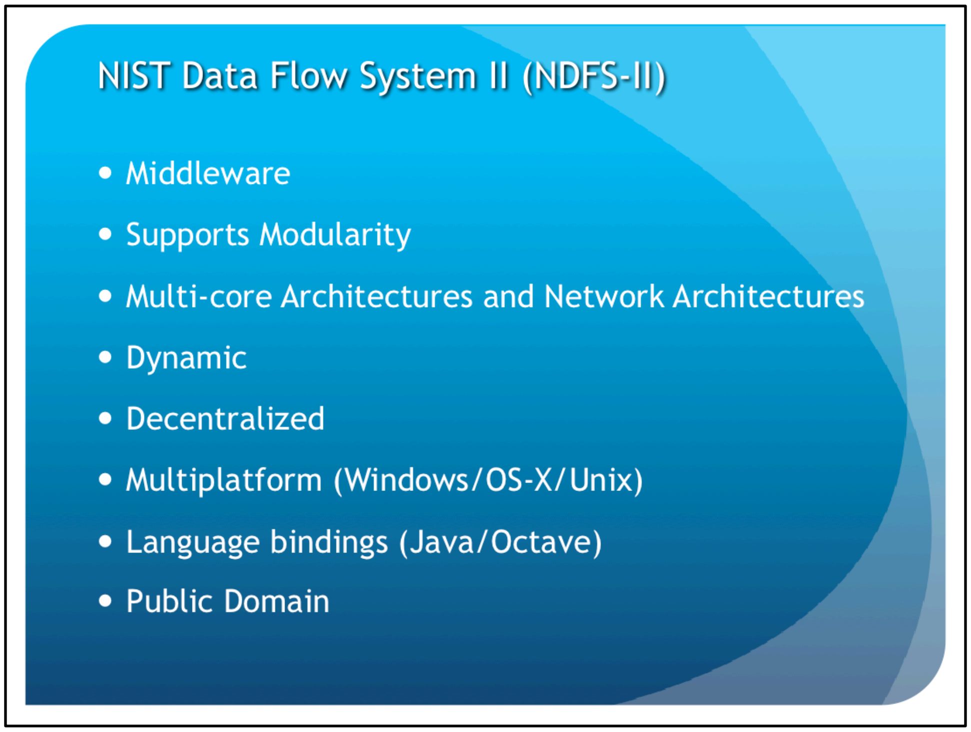
- Source, can be a sensor of any rate
- Filter, process and fusion
- Sink, process and fusion

Besides the distributed data transport and processing the filters and sinks can SPLIT and FUSE the data.

NIST Data Flow System II (NDFS-II)


Connection requests from Local Client Nodes
SAP
Client Acceptor Event Handler
Server Discovery Event Handler
Connection requests from Local Duplicators
Duplicator Acceptor Event Handler
Reactor
Connection requests from Remote Servers
Server Acceptor Event Handler
Server Event Handler
Server Connection Manager
Client Event Handler
Duplicator Event Handler
Server Event Handler
Reactor
Server
Duplicator
Client Node (P)
Flow Thread
Server Event Handler
Duplicator Event Handler
Client Event Handler
Connection requests from Local Client Nodes
Client Acceptor Event Handler
Reactor
Connection requests from Remote Duplicators
Duplicator Acceptor Event Handler
HOST A
Server
Client Node (C)
Read
Duplicator
Write
SHM
Read
Client Node (C)
HOST B
Service Access Point
Shared Memory
Direction of Messages
Data Queue
Registered to the Reactor
Shared Memory Access
(P) Producer
(C) Consumer

This is a concept picture of the NIST Data Flow System II. See Documentation on Website for details.



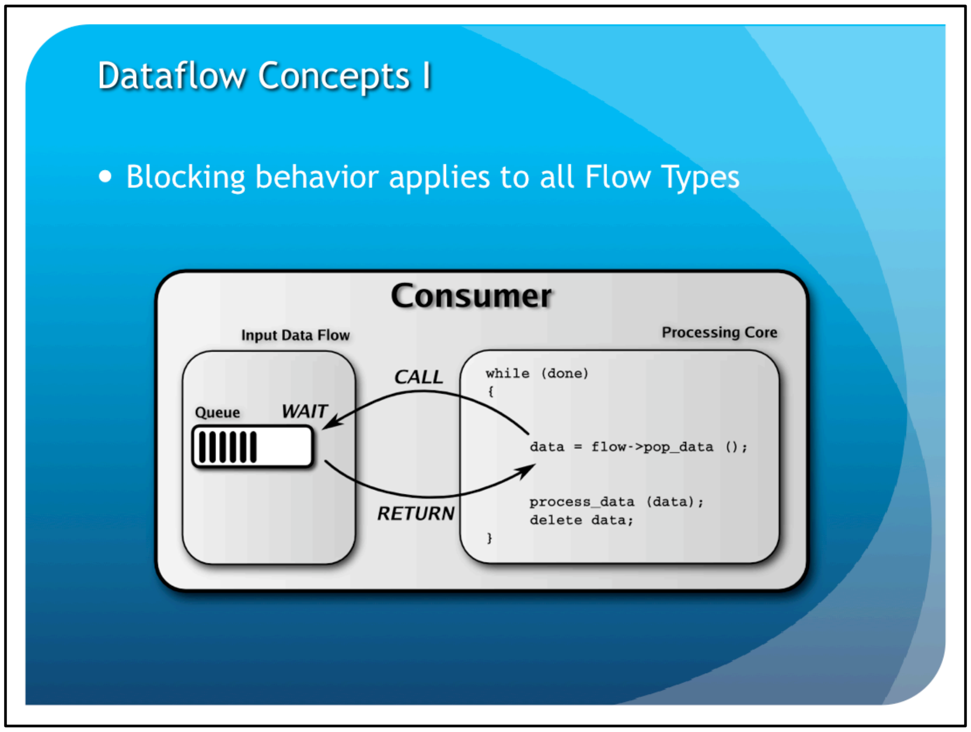
NIST Data Flow System II (NDFS-II)

- Middleware
- Supports Modularity
- Multi-core Architectures and Network Architectures
- Dynamic
- Decentralized
- Multiplatform (Windows/OS-X/Unix)
- Language bindings (Java/Octave)
- Public Domain

The NDFS-II is a sensor net middleware: a GENERIC transport vehicle

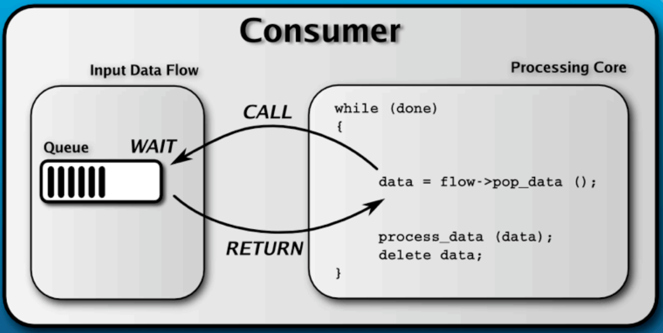
We do apply it momentarily for sensor data acquisition, transport, distributed processing, synchronization, storage, and retrieval of multimodal *corpora*. At this point it supports a variety of sensors for multimodal applications with simultaneous use of multiple OS platforms (e.g. Windows, OS X, and Linux).

See the NDFS-II Guide for a detailed description and documentation on our website (www.nist.gov/smartspace)



Dataflow Concepts I

- Blocking behavior applies to all Flow Types



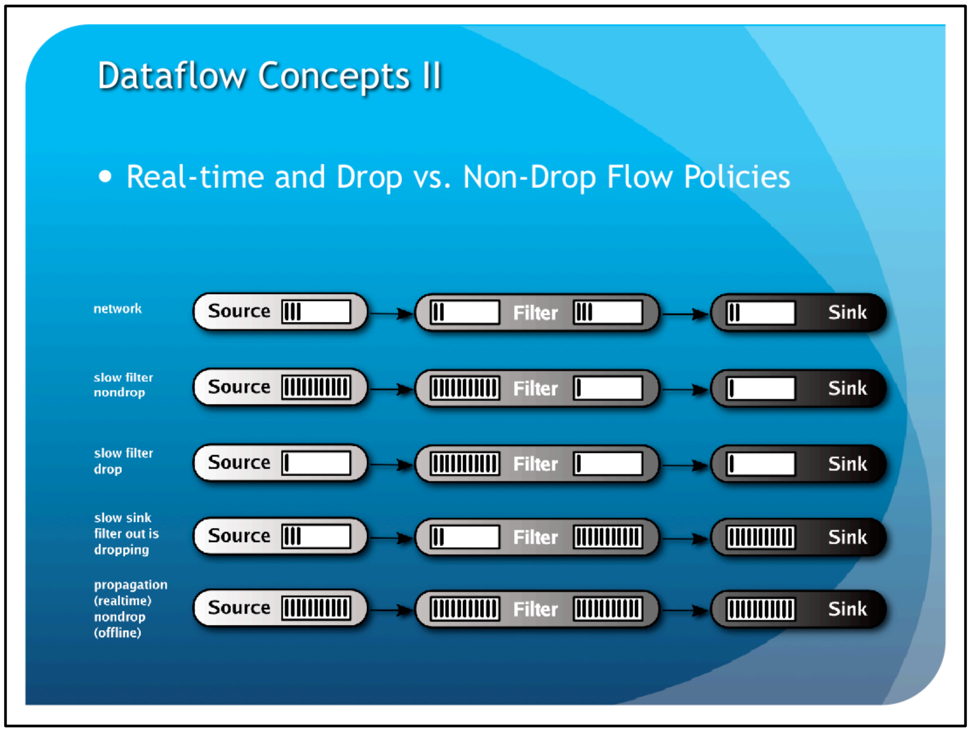
```
while (done)
{

    data = flow->pop_data ();

    process_data (data);
    delete data;
}
```

Basic behaviors of a client node:

In a processing loop of the client node the user requests data from the input queue or pushes data to an output queue (the so called flow). This call has a BLOCKING, which means if there is no data the user waits or if the output queue if full the user waits as well when pushing data. This behavior has important consequences on how the data is transported across several processing nodes.



Dataflow Concepts II

- Real-time and Drop vs. Non-Drop Flow Policies

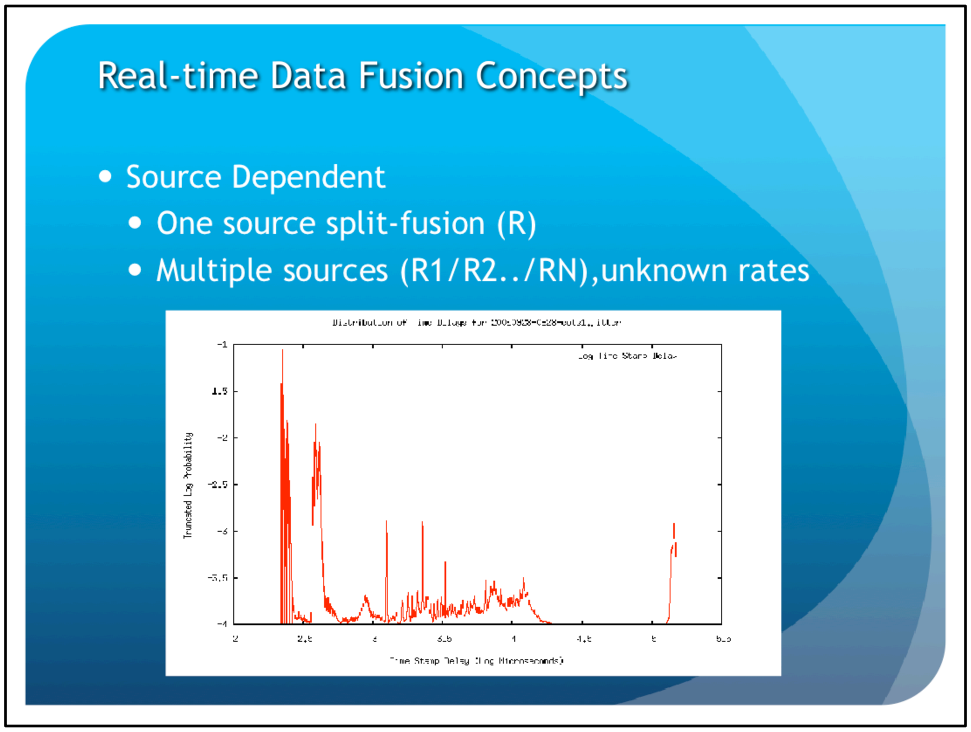
network
slow filter nondrop
slow filter drop
slow sink filter out is dropping
propagation (realtime) nondrop (offline)
Source
Filter
Sink

A second behavior is dropping of data. In order to keep a system running in real time on hardware which does not grant enough ressources (too slow processing of data) the user decides to drop data at the input or output flow level.

In online processing this has severe consequences:

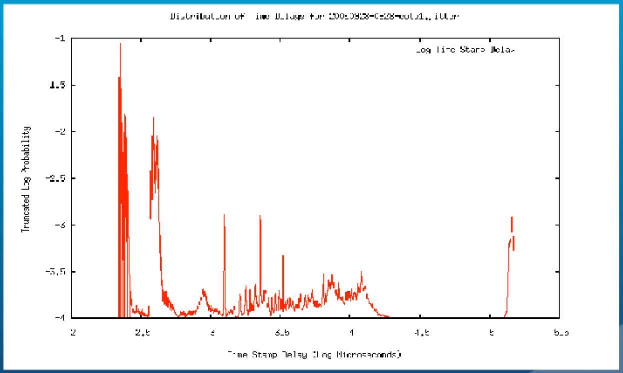
In the case of a slow filter, queues are filled up upto the source node, which itself starts CAPTURING at a lower rate (e.g. camera capture call occurs not so often anymore). This means that the source node lowers his capture rate. The consequence is that if another processing node is connected to the sources ouput flow, it will receive the data at a lower rate too. This behaviour is not desired. As a solution we adjust the filter node to drop the data. The source captures data at its full rate and only the FILTER drops the data if it can not process data in time. One can easily image that buffering inequalities (such as netwotk transport errors, uninterruptable kernel time, scheduling problems etc..) have an impact on the data flow, which especially impacts fused data.

Offline processing actually does not involve dropping of data, because the processing time is not critical.



Real-time Data Fusion Concepts

- Source Dependent
  - One source split-fusion (R)
  - Multiple sources (R1/R2../RN),unknown rates

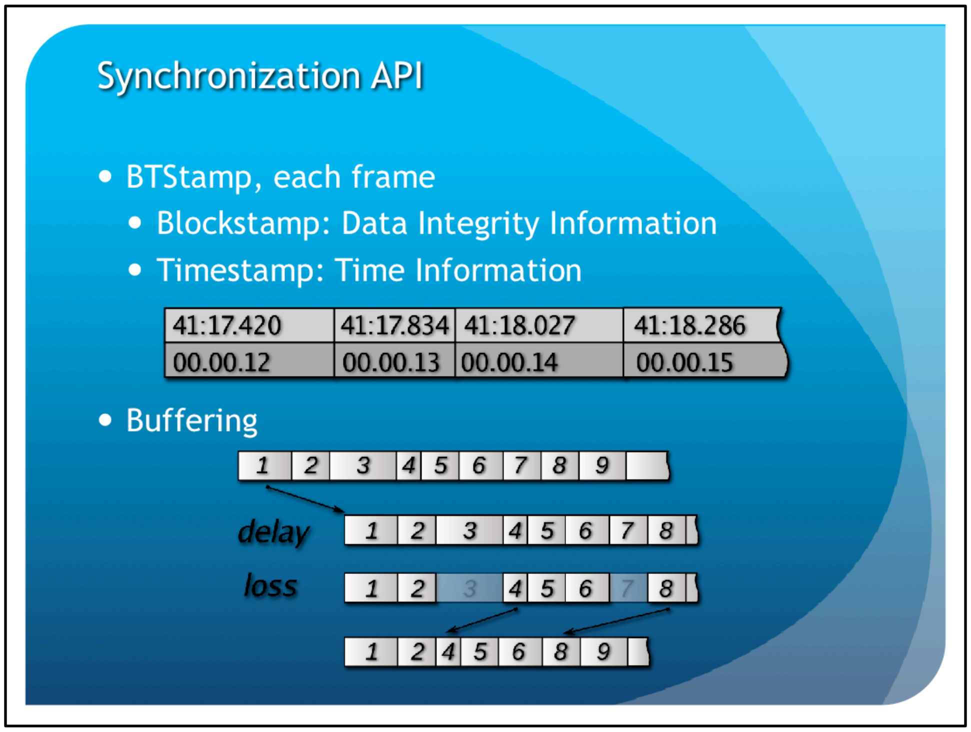


Two possible ways to fuse data:

Splitting data from a single source (distribute one signal) and fusing it back together.
Fusing data from many independent source with unknown rates.

Additionally to transport problems there is a jitter on the computer clock, so even if the sensor is giving us data at a constant rate, the os is introducing jitter. The graph shows a complex distributed jitter over several orders of magnitude.

Linear regression can be used to estimate the exact timestamp a posteriori (see references at the end about this)



Synchronization API

- BTStamp, each frame
  - Blockstamp: Data Integrity Information
  - Timestamp: Time Information

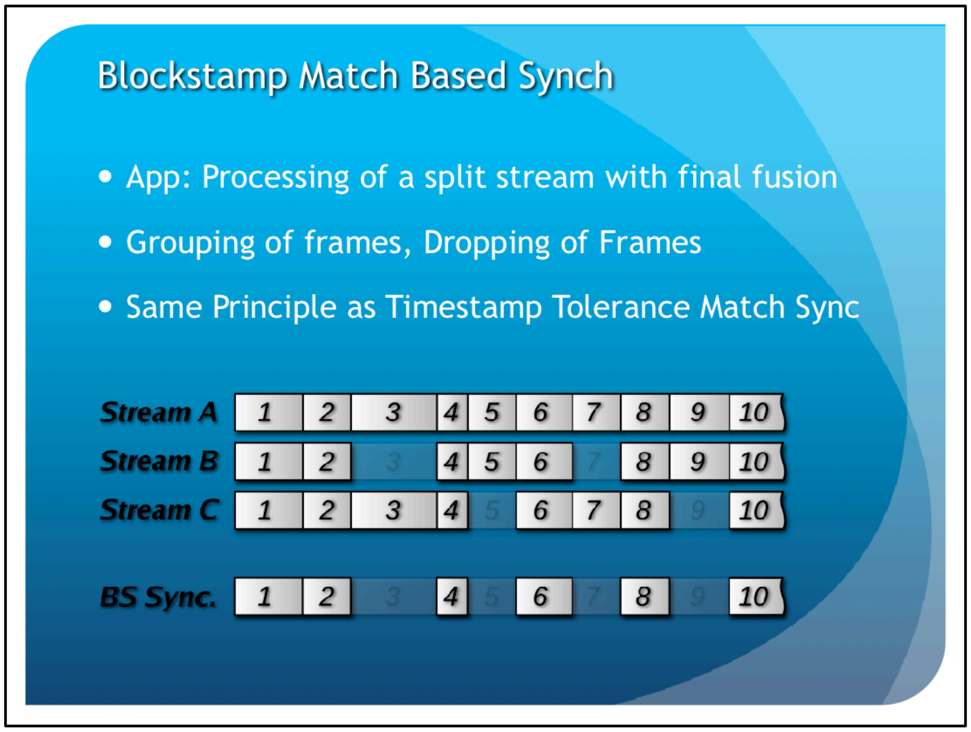
41:17.420	41:17.834	41:18.027	41:18.286
00.00.12	00.00.13	00.00.14	00.00.15

- Buffering

1 2 3 4 5 6 7 8 9
delay
1 2 3 4 5 6 7 8
loss
1 2 3 4 5 6 7 8
1 2 4 5 6 8 9

To be able to fuse data which involves a proper synchronization first we developed a SYNCHRONIZARTION API on top of the NDFS-II. The synchronization algorithm has multiple modes which will be explained in the next slides.

We introduce a mixture of Blockstamp and Timestamp as metadata. The blockstamps can be used to track data integrity and apply linear regression. Timestamps are used to synchronize data in time (The timestamps are produces by the os/computer clock and usually synchronized by NTP). Buffering has additionally to be applied to compensate for processing delay (due to transport besides the computation) and for data loss occurring due to dropping of data.



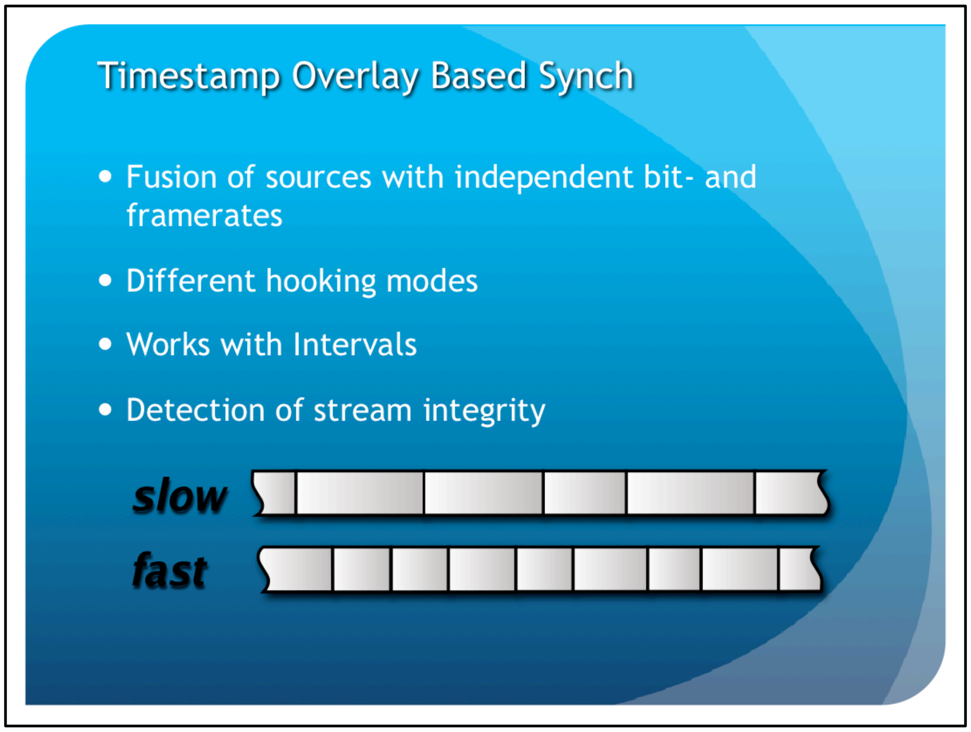
Blockstamp Match Based Synch

- App: Processing of a split stream with final fusion
- Grouping of frames, Dropping of Frames
- Same Principle as Timestamp Tolerance Match Sync

Stream A
1
2
3
4
5
6
7
8
9
10
Stream B
1
2
3
4
5
6
7
8
9
10
Stream C
1
2
3
4
5
6
7
8
9
10
BS Sync.
1
2
3
4
5
6
7
8
9
10

In the case of data originating only from a SINGLE sensor, distributed across multiple processing nodes and fused at the end, we can apply pure block-stamp based synchronization.

After compensation of the delay the data looks aligned as in the figure above (Stream A-B-C). The synchronization algorithm basically groups frames with matching blockstamps. Please keep in mind that we are in a distributed system so each stream has to be buffered due to time delay, so several Buffering methods can be used (adapting, fixed, unlimited).

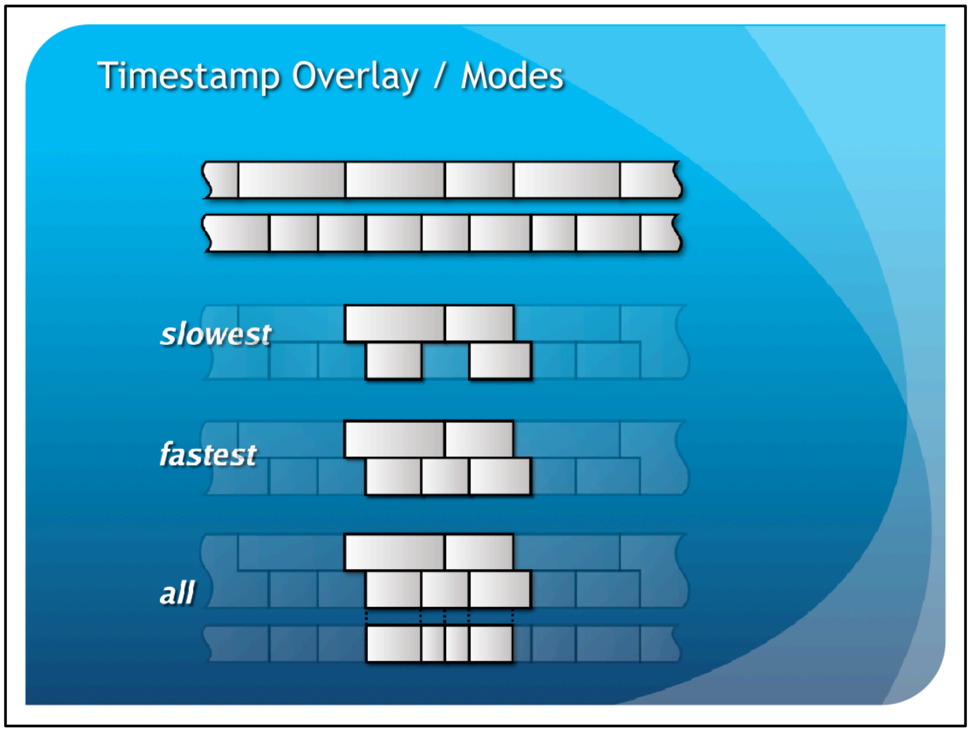


Timestamp Overlay Based Synch

- Fusion of sources with independent bit- and framerates
- Different hooking modes
- Works with Intervals
- Detection of stream integrity

Most interesting is the Timestamp Overlay Based Synchronization. This algorithm allows to fuse MULTIPLE sources with an UNKNOWN/INDEPENDENT data-rate. Several so called HOOKING modes exist (explained in the next slide). The algorithm works with time intervals gathers from the frames Timestamps instead of single timestamps itself. Due to the embedded Blockstamp in the BTStamp metadata, stream integrity can be additionally monitored.

Let’s assume two data stream from independent sources with different rates as an explanation example. (slow/fast)



Timestamp Overlay / Modes

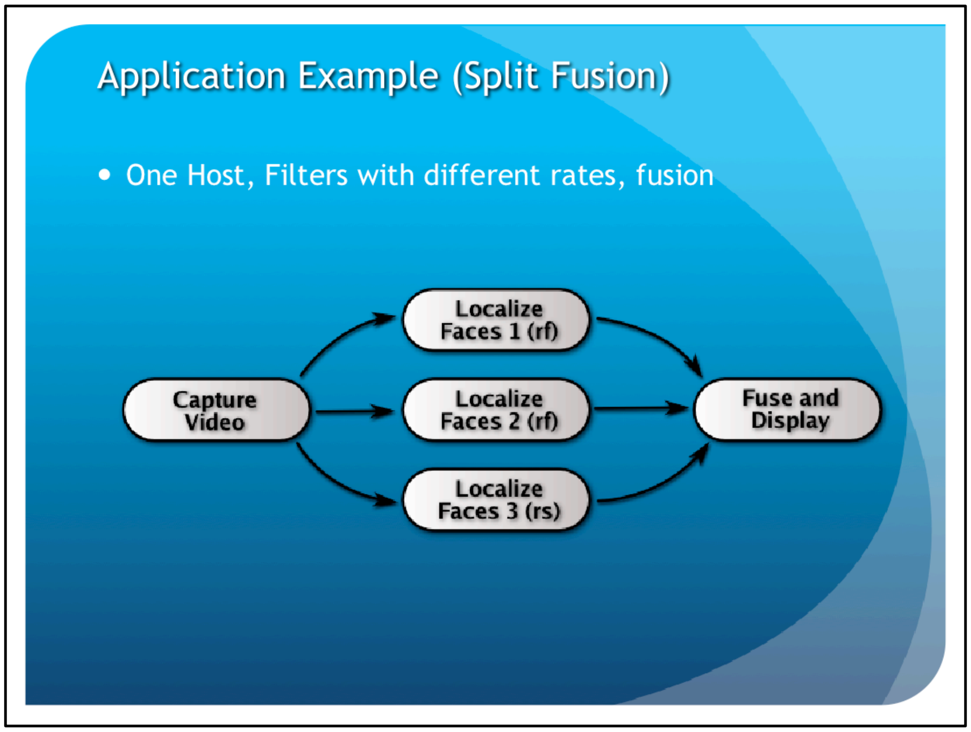
The three hooking modes allow different synchronization behavior:

All methods compensate for delay first (buffering).

Hooking to the slowest stream, analyzes the data-rate of all streams, determines the stream with the lowest rate and tries to group all frames which starting point falls into the interval of the slowest stream.

Hooking to the fastest stream, analyzes the data-rate of all streams, determines the stream with the highest rate and tries to group all frames which starting point falls into the interval of the fastest stream.

Overlay Hooking (all) groups all frames whose starting times fall within the current locked frame. This mode can have a HIGHER data-rate than the fastest stream. This is due to duplication of frames which can occur if the frames time-intervals interleave. The data integrity (duplication) can be determined however.



Application Example (Split Fusion)

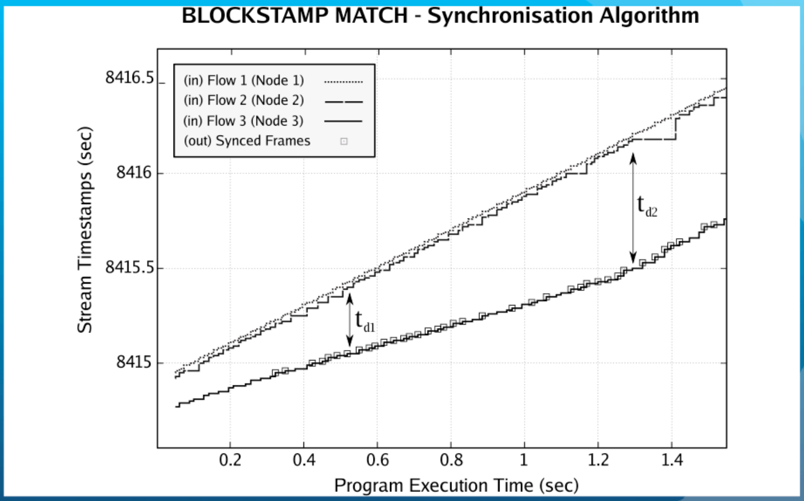
- One Host, Filters with different rates, fusion

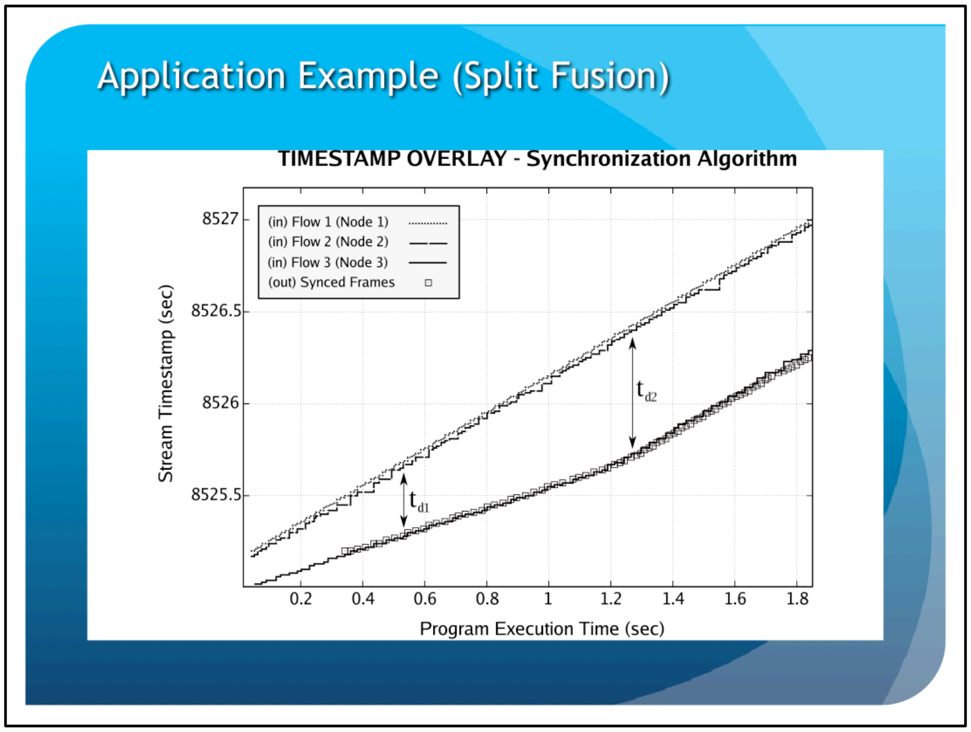
Capture Video
Localize Faces 1 (rf)
Localize Faces 2 (rf)
Localize Faces 3 (rs)
Fuse and Display

Application example as described in the corresponding ICME-08 paper. Please look up the paper for details.

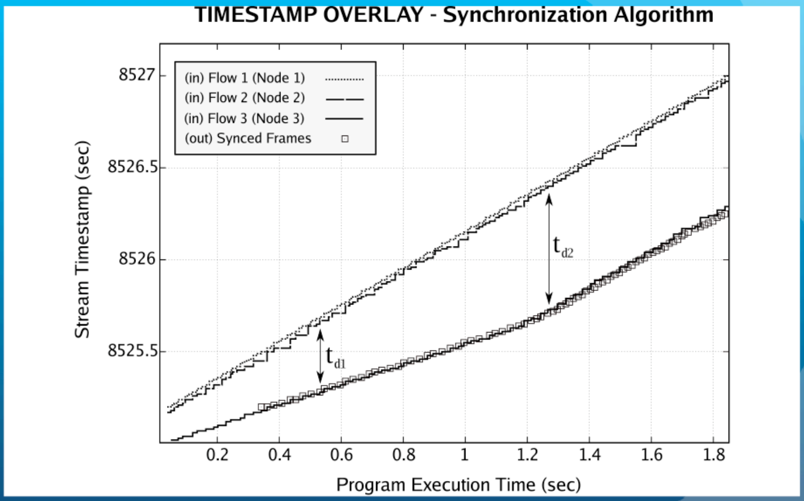
We show a split fusion example demonstrating runtime effects using the two explained synchronization methods.

Application Example (Split Fusion)

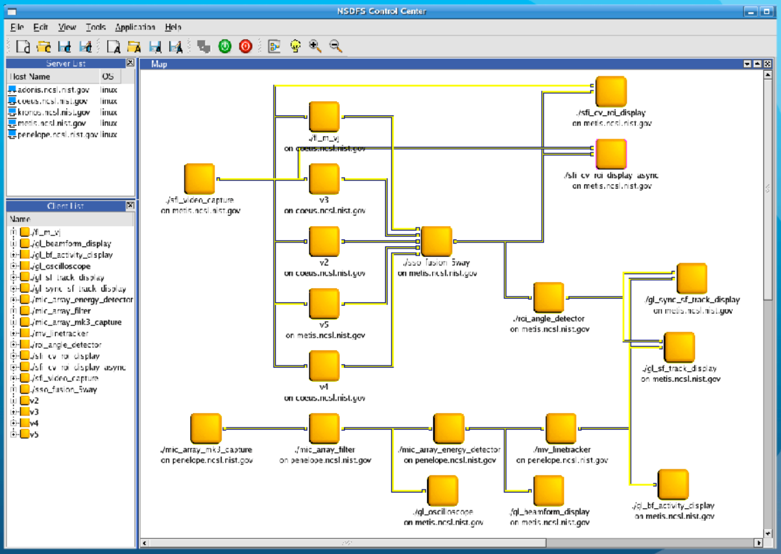




Application Example (Split Fusion)



Distributed Audio Visual Sensor Fusion - Map



NSDFS Control Center
File Edit View Tools Application Help
Server List
Map
Client List
./fl_m_vj
./gl_beamform_display
./gl_bf_activity_display
./gl_oscilloscope
./gl_sf_track_display
./gl_sync_sf_track_display
./mic_array_energy_detector
./mic_array_filter
./mic_array_mk3_capture
./mv_linetracker
./roi_angle_detector
./sfi_cv_roi_display
./sfi_cv_roi_display_async
./sfi_video_capture
./sso_fusion_5way
v2
v3
v4
v5
./sfi_cv_roi_display on metis.ncsl.nist.gov
./fl_m_vj on coeus.ncsl.nist.gov
./sfi_cv_roi_display_async on metis.ncsl.nist.gov
./sfi_video_capture on metis.ncsl.nist.gov
v3 on coeus.ncsl.nist.gov
v2 on coeus.ncsl.nist.gov
./sso_fusion_5way on metis.ncsl.nist.gov
v5 on metis.ncsl.nist.gov
v4 on coeus.ncsl.nist.gov
./roi_angle_detector on metis.ncsl.nist.gov
./gl_sync_sf_track_display on metis.ncsl.nist.gov
./gl_sf_track_display on metis.ncsl.nist.gov
./mic_array_mk3_capture on penelope.ncsl.nist.gov
./mic_array_filter on penelope.ncsl.nist.gov
./mic_array_energy_detector on penelope.ncsl.nist.gov
./mv_linetracker on penelope.ncsl.nist.gov
./gl_oscilloscope on metis.ncsl.nist.gov
./gl_beamform_display on metis.ncsl.nist.gov
./gl_bf_activity_display on metis.ncsl.nist.gov

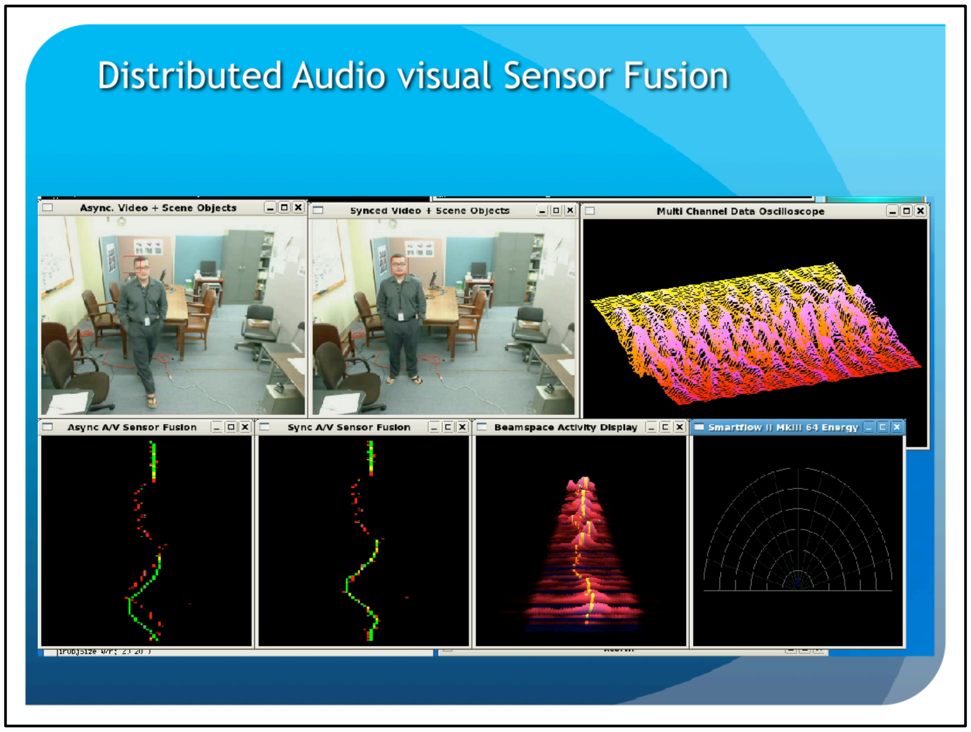
Host Name	OS
adonis.ncsl.nist.gov	linux
coeus.ncsl.nist.gov	linux
kronos.ncsl.nist.gov	linux
metis.ncsl.nist.gov	linux
penelope.ncsl.nist.gov	linux

Another example of a real-time audio visual sensor fusion application, applying all described forms of SYNCHRONIZATION at the fusion steps.

Two independent sensors (video camera, 64 channel microphone array) are processed and monitored in a separate processing pipleline:

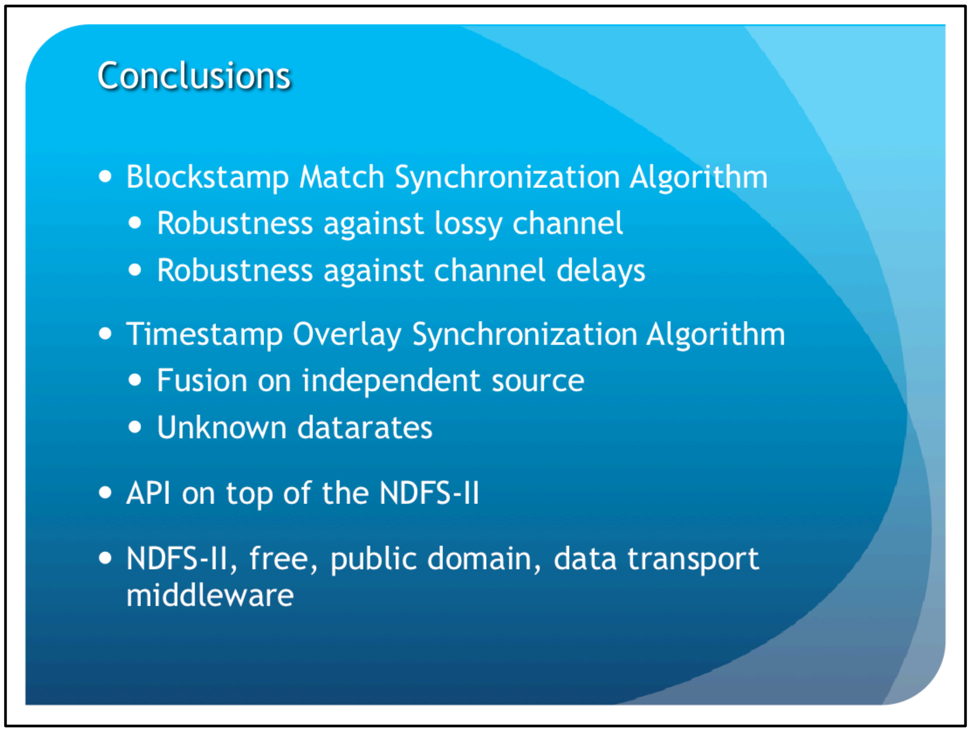
Video stream is split in 5 processing nodes to perform a face localization. The ROIS are fused afterwards using Blockstamp based sychronization. Both the video stream and the ROI stream a displayed either asynchronous (as the data comes into the sink) or synchronous.

Multichannel Audio stream is at first filtered spectrally and spatially, energy detection (bearing angle) is performed using a beamformer. A linetracker and speech activity detector is used afterwards to keep track of the speaker and assign the speakers angle. Same assignment is performed on the ROI stream. In the end the both angle feature streams are fused either asynchronous or synchronous to demonstrate the error occuring due to processing and transportation delay, as well as due to different rates.


Distributed Audio visual Sensor Fusion
Async. Video + Scene Objects
Synced Video + Scene Objects
Multi Channel Data Oscilloscope
Async A/V Sensor Fusion
Sync A/V Sensor Fusion
Beamspace Activity Display
Smartflow II MkIII 64 Energy

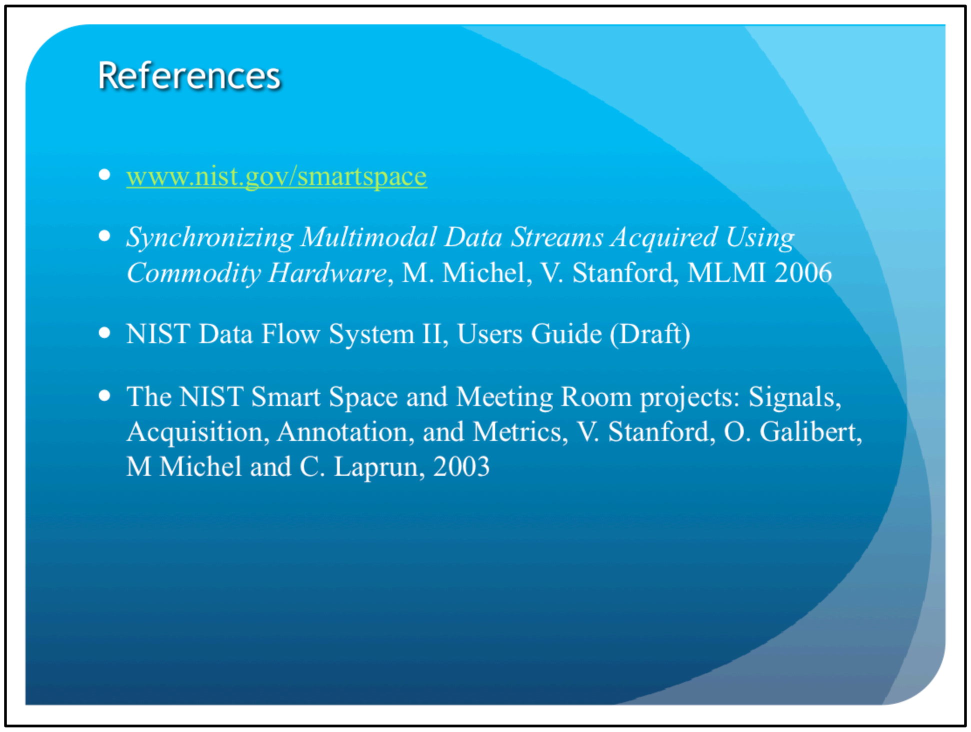
From left/top to right/bottom:

-asynchronous real-time Video/ROI display, ROI is off due to delayed incoming data
- synchronous real-time Video/ROI display, ROI and video match
- 64 channel real-time audio display
- asynchronous real-time Audio Visual Track display, tracks should overlay but due to drift and different rates the tracks show a fusion error
- synchronous real-time Audio Visual Track display, tracks overlay due to 'timestamp overlay' synch. Method
- 3d real-time Audio Track visualization of the activity/linetracking module
- Real-time bearing Angle, Acoustic Energy monitor



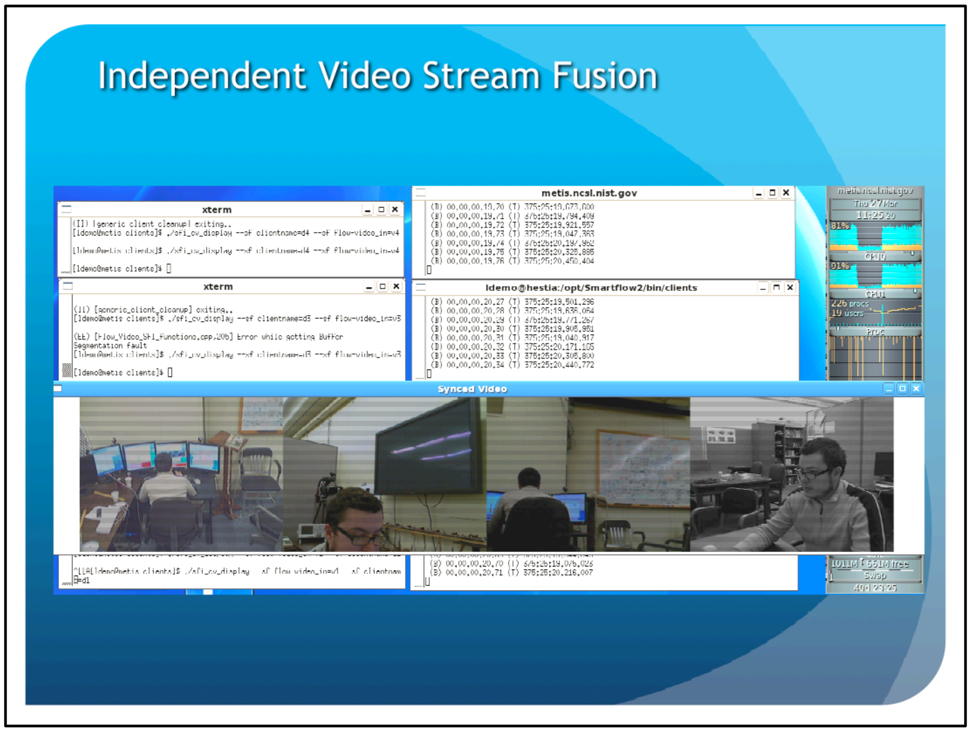
Conclusions

- Blockstamp Match Synchronization Algorithm
  - Robustness against lossy channel
  - Robustness against channel delays
- Timestamp Overlay Synchronization Algorithm
  - Fusion on independent source
  - Unknown datarates
- API on top of the NDFS-II
- NDFS-II, free, public domain, data transport middleware



References

- www.nist.gov/smartspace
- *Synchronizing Multimodal Data Streams Acquired Using Commodity Hardware*, M. Michel, V. Stanford, MLMI 2006
- NIST Data Flow System II, Users Guide (Draft)
- The NIST Smart Space and Meeting Room projects: Signals, Acquisition, Annotation, and Metrics, V. Stanford, O. Galibert, M Michel and C. Laprun, 2003



Independent Video Stream Fusion

xterm

```
(II) [generic client cleanup] exiting..
[ldemo@metis clients]$ ./sfi_cv_display --sf clientname=d4 --sf flow-video_in=v4

[ldemo@metis clients]$ ./sfi_cv_display --sf clientname=d4 --sf flow-video_in=v4

[ldemo@metis clients]$
```

xterm

```
(II) [generic_client_cleanup] exiting..
[ldemo@metis clients]$ ./sfi_cv_display --sf clientname=d3 --sf flow-video_in=v3

(EE) [Flow_Video_SFI_functions.cpp,206] Error while getting Buffer
Segmentation fault
[ldemo@metis clients]$ ./sfi_cv_display --sf clientname=d3 --sf flow-video_in=v3

[ldemo@metis clients]$
```

```
^[[A[ldemo@metis clients]$ ./sfi_cv_display --sf flow-video_in=v1 --sf clientnam
e=d1
```

metis.ncsl.nist.gov

```
(B) 00,00,00,19,70 (T) 375:25:19,673,600
(B) 00,00,00,19,71 (T) 375:25:19,794,409
(B) 00,00,00,19,72 (T) 375:25:19,921,557
(B) 00,00,00,19,73 (T) 375:25:19,047,363
(B) 00,00,00,19,74 (T) 375:25:20,197,962
(B) 00,00,00,19,75 (T) 375:25:20,325,885
(B) 00,00,00,19,76 (T) 375:25:20,450,404
```

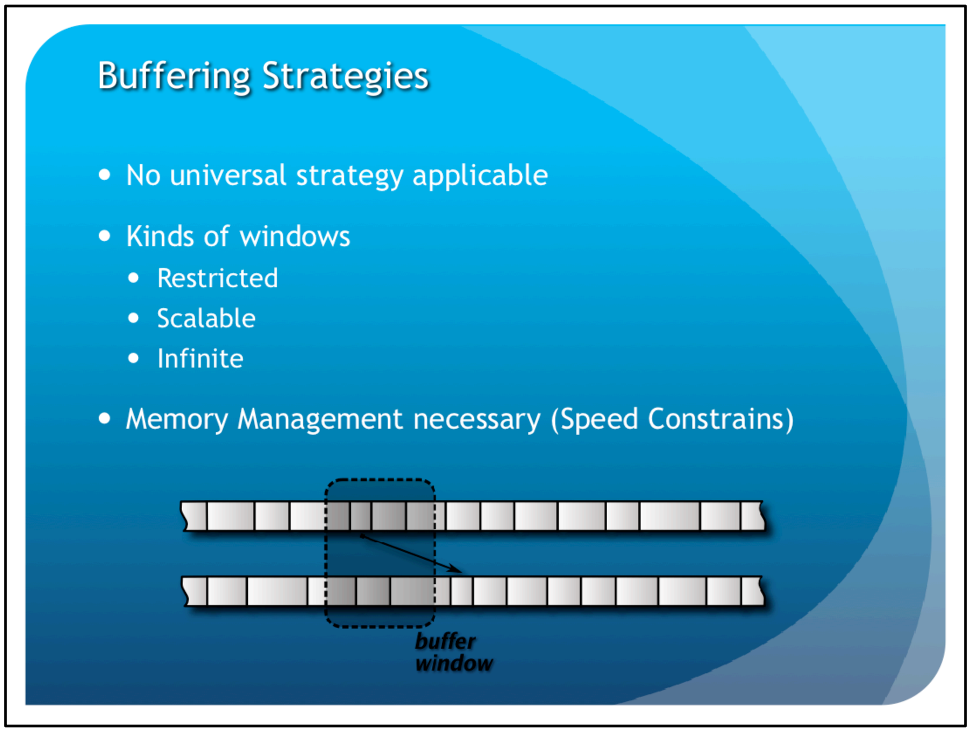
ldemo@hestia:/opt/Smartflow2/bin/clients

```
(B) 00,00,00,20,27 (T) 375:25:19,501,296
(B) 00,00,00,20,28 (T) 375:25:19,636,064
(B) 00,00,00,20,29 (T) 375:25:19,771,267
(B) 00,00,00,20,30 (T) 375:25:19,905,951
(B) 00,00,00,20,31 (T) 375:25:19,040,917
(B) 00,00,00,20,32 (T) 375:25:20,171,105
(B) 00,00,00,20,33 (T) 375:25:20,305,800
(B) 00,00,00,20,34 (T) 375:25:20,440,772
```

```
(B) 00,00,00,20,70 (T) 375:25:19,075,028
(B) 00,00,00,20,71 (T) 375:25:20,216,007
```

Additional slide:

example of 4 independent camera stream fusion



Buffering Strategies

- No universal strategy applicable
- Kinds of windows
  - Restricted
  - Scalable
  - Infinite
- Memory Management necessary (Speed Constrains)

buffer
window

Additional Slide:

Need of windowing due to delay. Windowing implies memory usage and have to be chosen properly