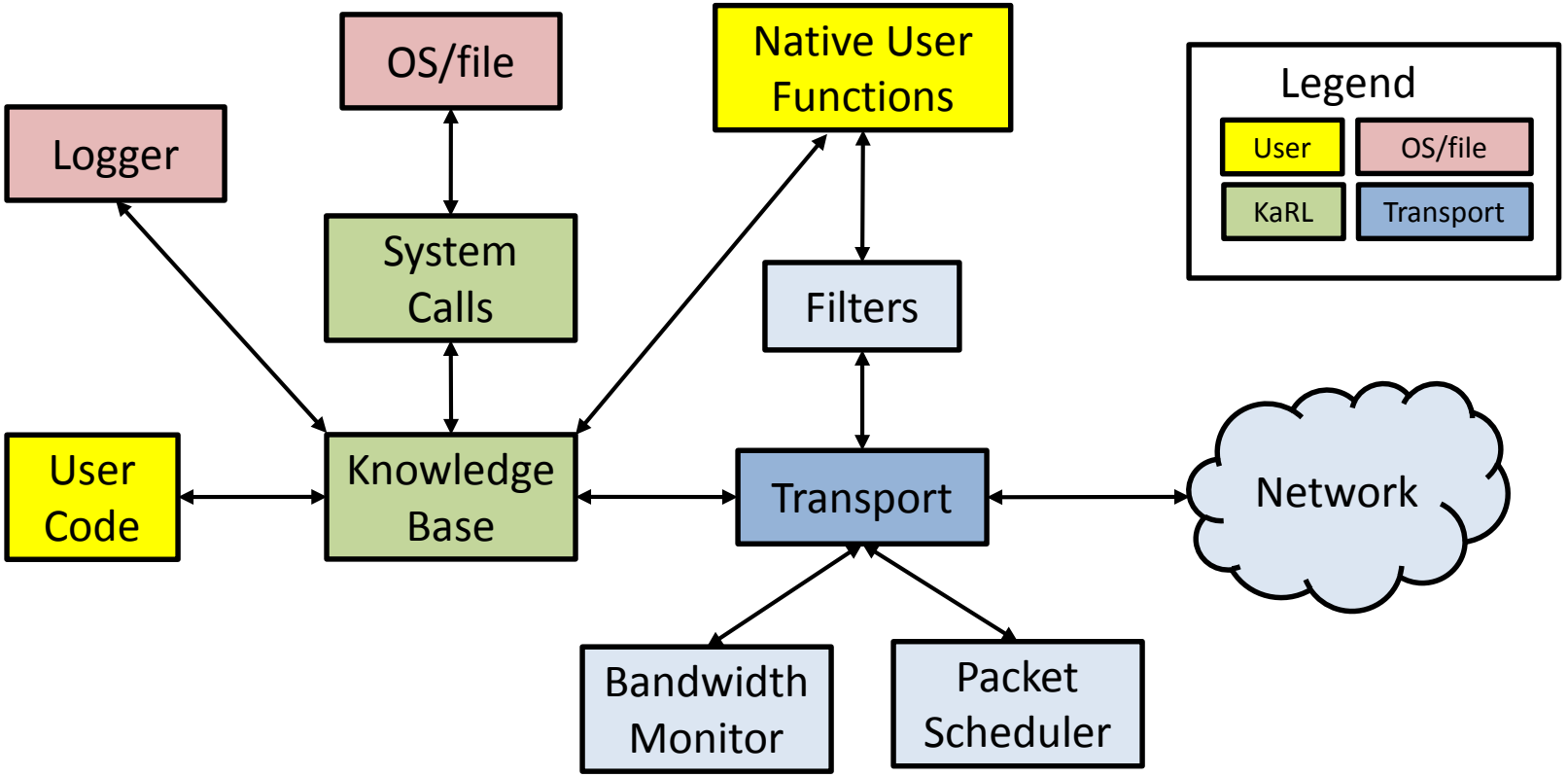
MADARA: An Open Architecture for Collaboration, Timing and Control

James Edmondson

<jedmondson@gmail.com>

About MADARA: What is it?

MADARA is a high performance middleware and toolkit that enables rapid prototyping of distributed applications, especially soft real-time systems



About MADARA: How is it different?

- Nanosecond execution times through a focus on constant time operations
- Flexibility to integrate user callbacks on receive, send and rebroadcast
- Focus on Quality-of-Service, OS interactions, and control and timing
- Portable to various operating systems and architectures (ARM, Intel, AMD, Windows, Linux, Mac, Android, iPhone, etc.)
- User-defined filters for augmenting information (image shaping, UML/XMLification, packet dropping, etc.)
- First class support for strings, integers, doubles, arrays, and text and binary files
- First class support for UDP, Multicast, Broadcast, and DDS transports
- Bandwidth shaping, deadline filtering, transport monitoring to prevent overpublishing
- Extensibility to new transports, logical flows, and runtime code execution (controllable by the developer)
- Decentralized but allows for implementing centralized patterns like client/server, pub/sub, etc.
- Completely open source under a BSD license
- Well-documented in Wikis and Doxygen documentation

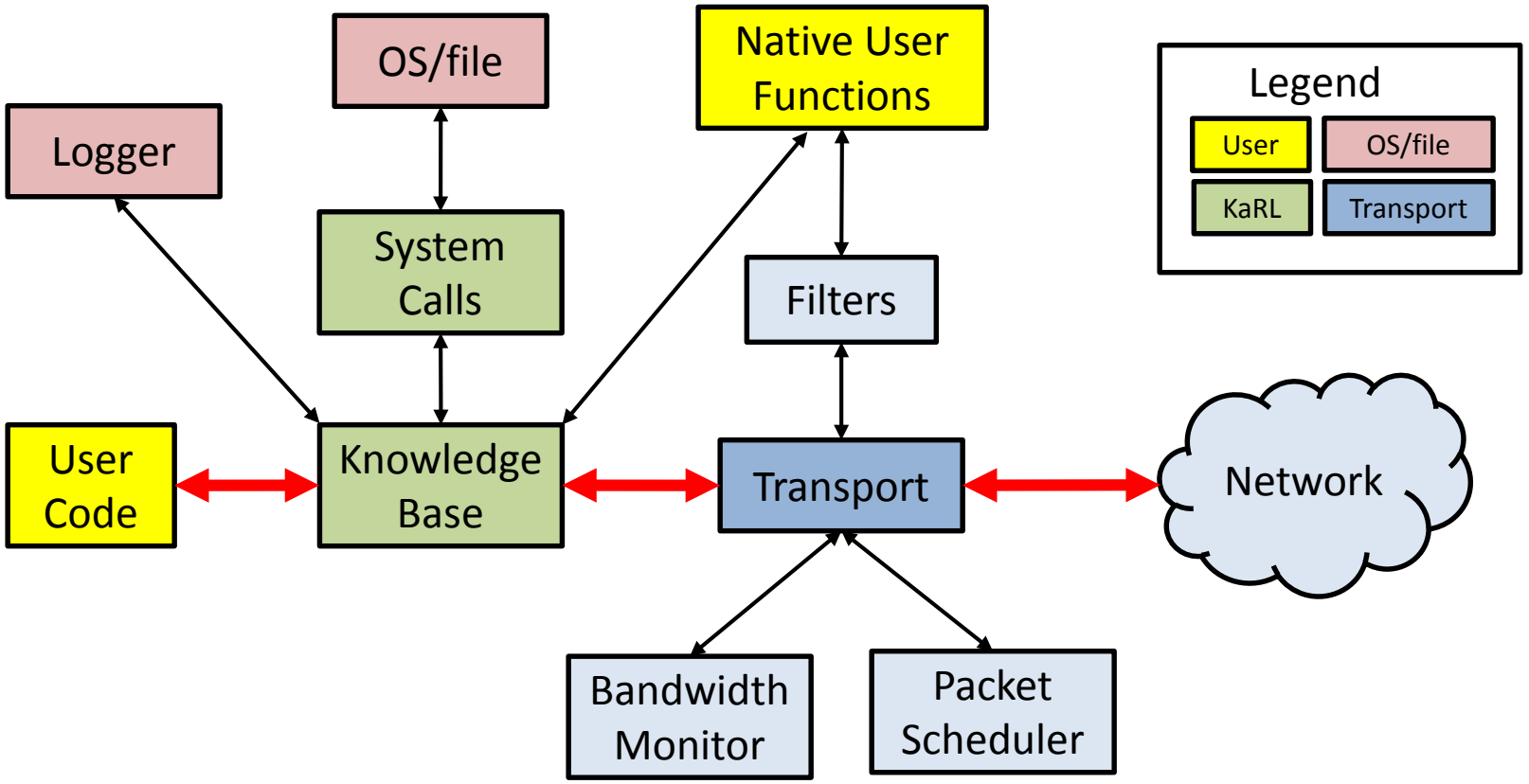
Target Usage for MADARA Development

- Rapid prototyping of distributed applications
- Any developers interested in using a scripting-language-like helper language for real-time system development
- Teachers who want to rapidly prototype advanced operating system concepts, even on-the-fly in front of students
- Anyone who needs portable middleware that specializes in nanosecond execution times, quality-of-service and networking support for usage on Android, Mac, Windows, Linux, etc.
- Who we are not really targeting: website developers

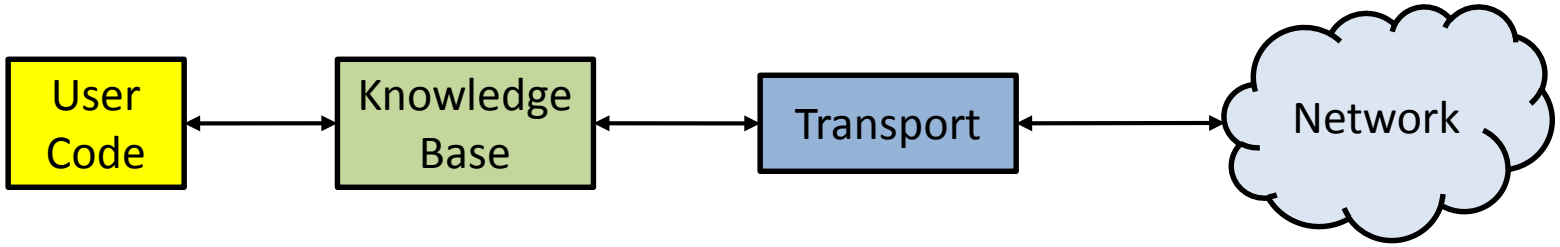
Example Usage of MADARA

SMASH Project at Carnegie Mellon University

First steps: Networking Basics



First steps: Networking Basics

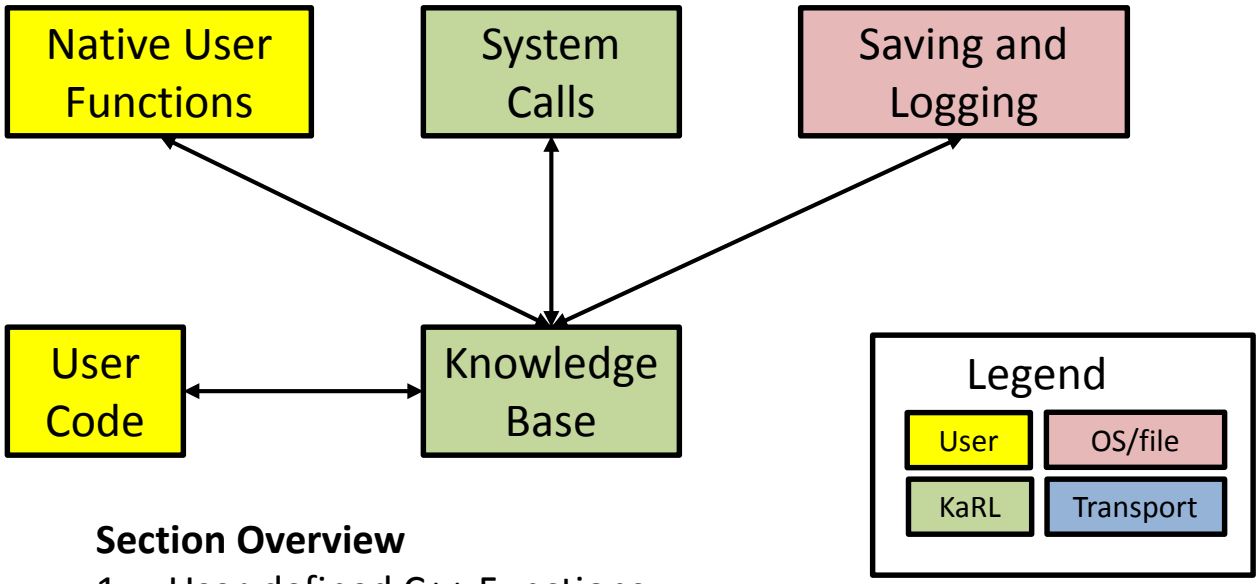


1. Include Knowledge Base
2. Setup Network Transport
3. Setup Knowledge Base
4. Query and Change the Context

Example code

```
#include "madara/knowledge_engine/Knowledge_Base.h"

Madara::Knowledge_Record::Integer my_id (1);

int main (int argc, char ** argv)
{
  Madara::Transport::QoS_Transport_Settings settings;
  settings.hosts.push_back ("239.255.0.1:4150");
  settings.type = Madara::Transport::MULTICAST;

  Madara::Knowledge_Engine::Eval_Settings eval_settings;

  Madara::Knowledge_Engine::Knowledge_Base knowledge ("", settings);

  Madara::Knowledge_Record processes = knowledge.get ("processes.deployed");
  knowledge.set (".id", my_id);
  knowledge.set ("process{.id}.ready", 1.0, eval_settings);

}
```

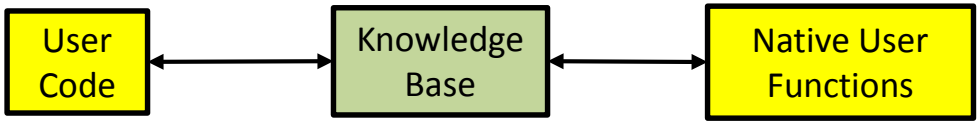
Intermediate Steps



Section Overview

1. User-defined C++ Functions
2. System Calls within MADARA
3. Saving and Loading Contexts

Intermediate Steps: Native User Functions



1. Function Arguments are referenced via a C++ vector or Java array

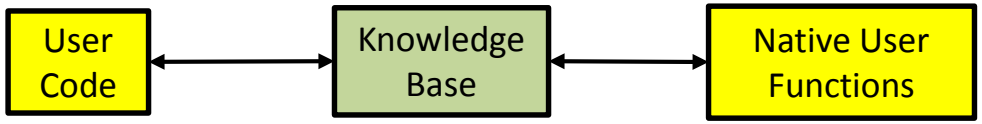
2. The Knowledge Base is accessible via the Variables Facade

3. Device drivers, external library calls, etc. can be called from within these function calls

Example of Creating External Native User Function Calls

```
Madara::Knowledge_Engine::Function_Arguments & args
                    Madara::Knowledge_Engine::Variables & vars)
{
Madara::Knowledge_Record::Integer num_objects = sense_objects ();
double temperature = poll_temperature ();
  vars.set (".sensed_objects", num_objects);
  vars.set (".temperature", temperature);

  return Madara::Knowledge_Record::Integer (1);
}

Madara::Knowledge_Record
react_to_environment (Madara::Knowledge_Engine::Function_Arguments & args,
Madara::Knowledge_Engine::Variables & vars
  if (vars.evaluate (".sensed_objects > 5 || .temperature > 100").is_true ())
    stop ();
  else
    move_to (next_target);

  return Madara::Knowledge_Record::Integer (1);
}
```

Intermediate Steps: Native User Functions



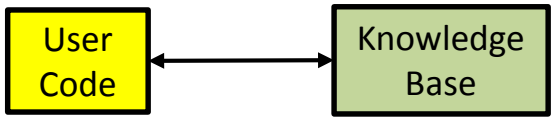
4. Define a named function in MADARA for the native user functions

5. Call the functions from within an evaluate or wait call

Example of Calling Native User Functions

```
…

int main (int argc, char ** argv)
{
  Madara::Knowledge_Engine::Knowledge_Base knowledge;

knowledge.define_function ("sense", sense_environment);
knowledge.define_function ("react", react_to_environment);

  while (knowledge.get ("terminated").is_false ())

knowledge.evaluate ("sense (); react ()");

  }
}
```

Intermediate Steps: Creating a Periodic Loop

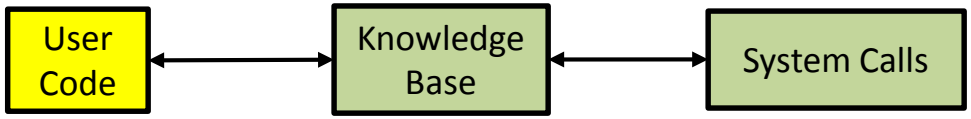


1. Define poll_frequency and max_wait_time inside of a wait settings class
2. Use a wait statement with the defined wait settings

Example of Calling Native User Functions

```
…

int main (int argc, char ** argv)
{
  Madara::Knowledge_Engine::Knowledge_Base knowledge;

  knowledge.define_function ("sense", sense_environment);
  knowledge.define_function ("react", react_to_environment);

Madara::Knowledge_Engine::Wait_Settings wait_settings;
wait_settings.poll_frequency = 0.050;  // every 50ms
wait_settings.max_wait_time = 100;     // 100s (-1 is infinite wait)

knowledge.wait ("!terminated => (sense (); react ())", wait_settings);
}
```

Intermediate Steps: Using System Calls

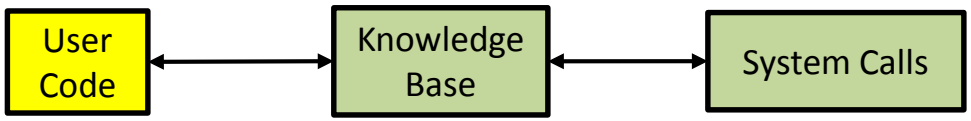


- System calls begin with a # and cannot be overridden by users.
- #print_system_calls will print all system calls along with documentation
- Most system calls map to functions you can call directly on the knowledge base or a knowledge record

Example of Using System Calls

```
…

int main (int argc, char ** argv)
{
  Madara::Knowledge_Engine::Knowledge_Base knowledge;

  knowledge.evaluate (
    ".begin_time = #get_time ();" // in nanoseconds
    ".file = #read_file ('\files\my_file.txt');"
    ".end_time = #get_time ();"
    ".total_time = .end_time - .begin_time;"
    ".file_size = #size (.file);"
    "#print ('Read {.file_size} bytes in {.total_time} ns.\n')"
  );
}
```

Intermediate Steps: Saving and Loading Contexts



- To save all variables in the context, use the save_context function
- To load all variables from a file, use the load_context function
- There is also a save_checkpoint function for incremental updates

Example of Saving and Loading Context

```
int main (int argc, char ** argv)
{
  Madara::Knowledge_Engine::Knowledge_Base knowledge;

  knowledge.evaluate (
    "begin_time = #get_time ();" // in nanoseconds
    "file = #read_file ('\files\my_file.txt');"
    "end_time = #get_time ();"
    "total_time = end_time - begin_time;"
    "file_size = #size (.file);"
    "clock = #clock ();"
  );

  Madara::Knowledge_Record clock = knowledge.get ("clock");
  std::string filename = "\files\my_context " + clock.to_string () + ".kbb";
```

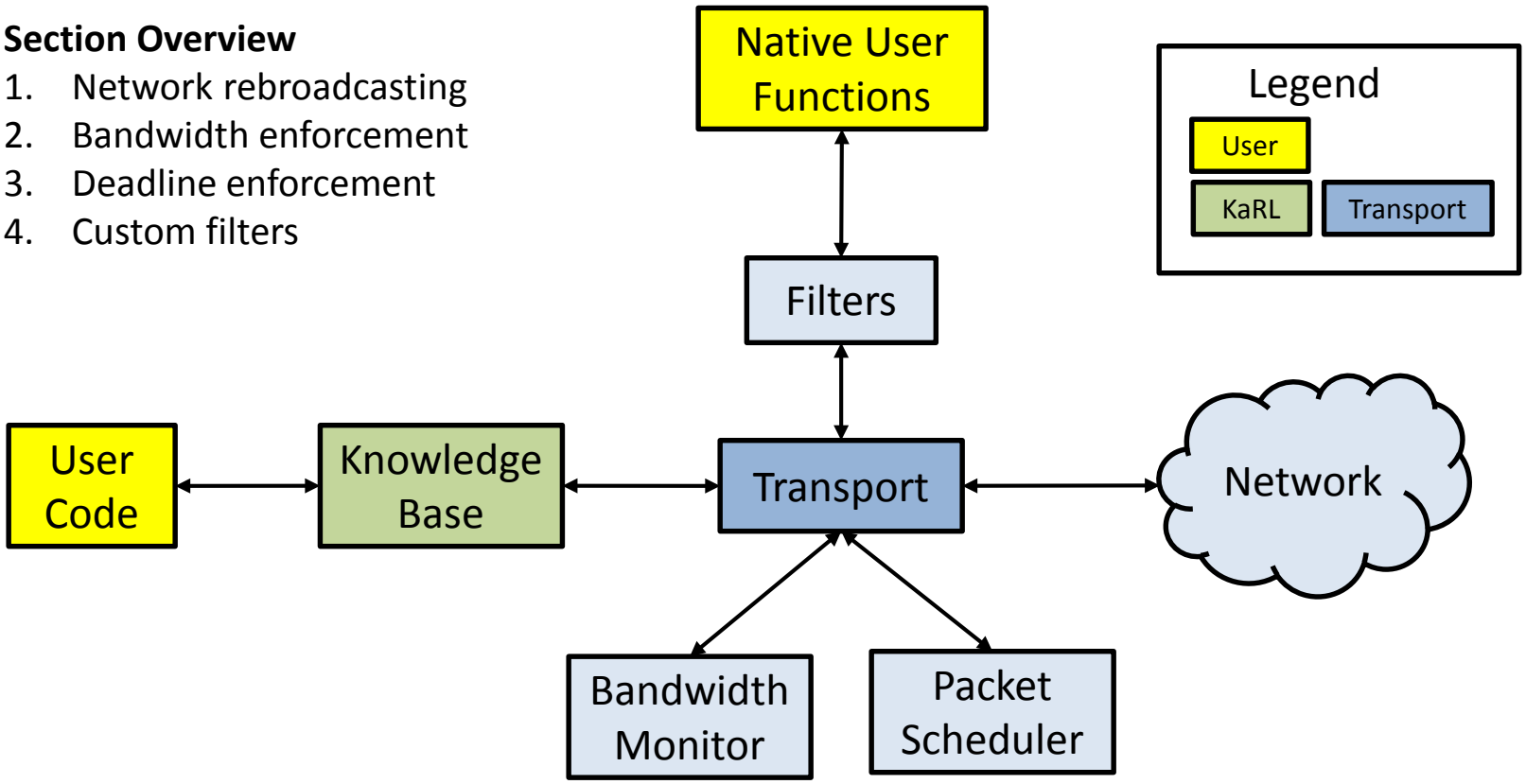
```
knowledge.save_context (filename);
```

```
knowledge.load_context (filename, false);
```

Advanced MADARA Features

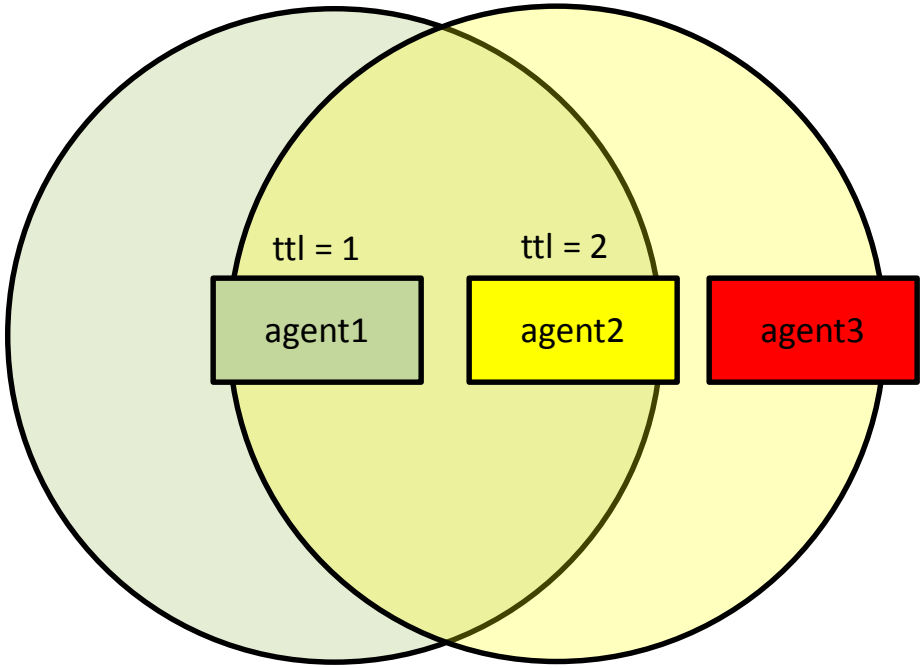
Section Overview

1. Network rebroadcasting
2. Bandwidth enforcement
3. Deadline enforcement
4. Custom filters



Advanced MADARA Features: Network Rebroadcasting

In wireless systems or connected networks, messages sometimes have to be routed to their intended targets

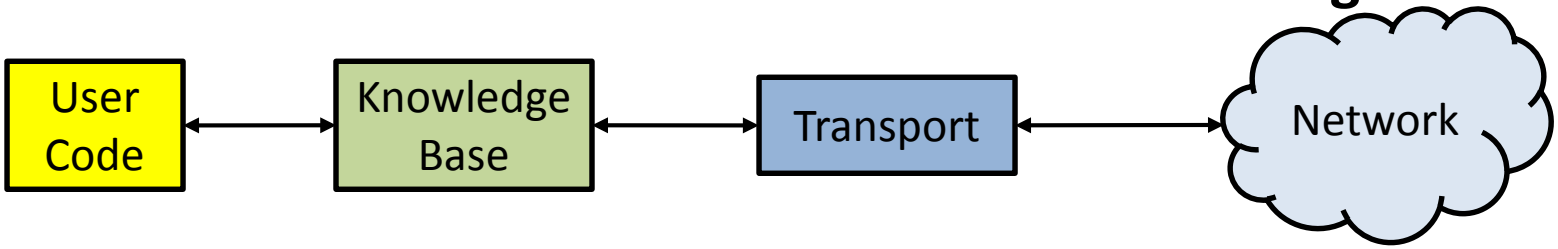


Example of agent1 sending with ttl of 1 and 2

In MADARA, we facilitate such network routing via the rebroadcast ttl feature.

This feature is also useful in unreliable networks where resends are necessary

Advanced MADARA Features: Network Rebroadcasting



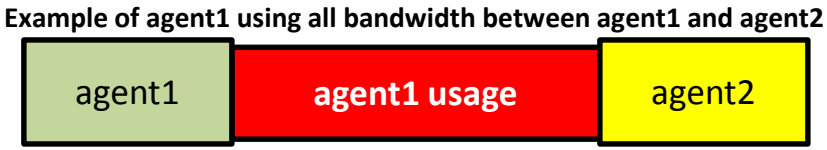
1. Setup normal transport settings
2. To participate in rebroadcasts, use the enable_participant_ttl function
3. To set the time-to-live (ttl) for rebroadcasting data from this agent, set the rebroadcast ttl.
4. Any updated global variables will be rebroadcasted by other agents

Example code

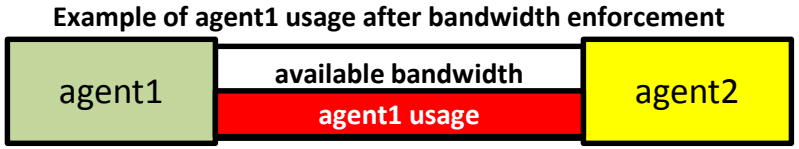
```
#include "madara/knowledge_engine/Knowledge_Base.h"

Madara::Knowledge_Record::Integer my_id (1);

int main (int argc, char ** argv)
{
  Madara::Transport::QoS_Transport_Settings settings;
  settings.hosts.push_back ("239.255.0.1:4150");
  settings.type = Madara::Transport::MULTICAST;

  settings.enable_participant_ttl ();

  settings.set_rebroadcast_ttl (2);

  Madara::Knowledge_Engine::Eval_Settings eval_settings;

  Madara::Knowledge_Record processes = knowledge.get ("processes.deployed");
  knowledge.set (".id", my_id);

  knowledge.set ("process{.id}.ready", 1.0, eval_settings);
}
```

Advanced MADARA Features: Bandwidth Enforcement

In real networks, bandwidth is finite and valuable, yet one publisher can overwhelm the entire network

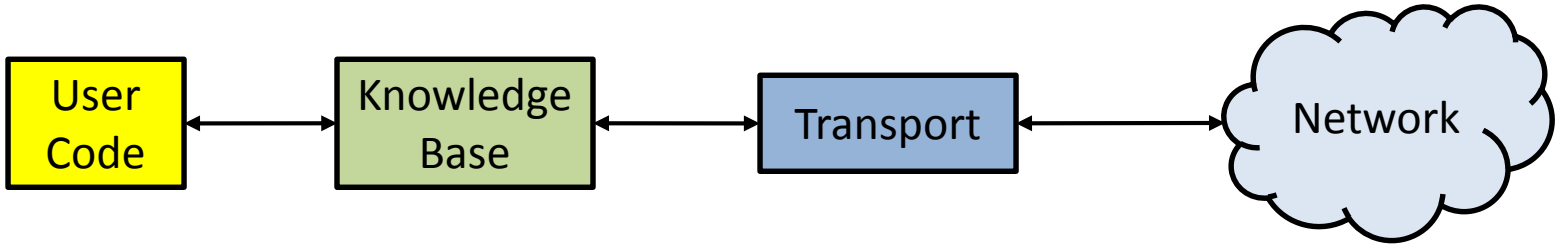


In MADARA, we facilitate provide both coarse-grained and fine-grained bandwidth enforcement. These features are very useful, especially when reading files into the knowledge base.



We next discuss coarse-grained bandwidth enforcement.

Advanced MADARA Features: Bandwidth Enforcement



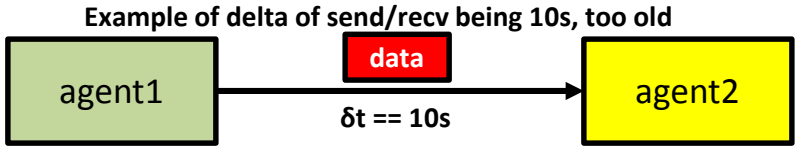
1. Setup normal transport settings
2. Send bandwidth limit regulates an upper limit on what this agent sends
3. Total bandwidth limit regulates sending based on what has been received over past 10s
4. No new updates will be sent unless the bandwidth usage is less than the limits

Example code

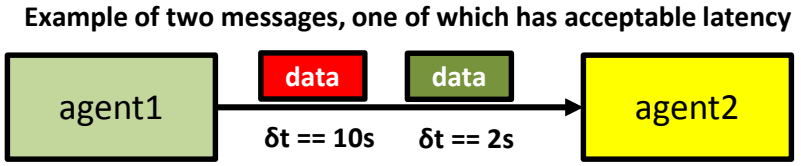
```
#include "madara/knowledge_engine/Knowledge_Base.h"

Madara::Knowledge_Record::Integer my_id (1);

int main (int argc, char ** argv)
{
  Madara::Transport::QoS_Transport_Settings settings;
  settings.hosts.push_back ("239.255.0.1:4150");
  settings.type = Madara::Transport::MULTICAST;
  settings.set_send_bandwidth_limit (100000); // 100KB/s over 10s
  settings.set_total_bandwidth_limit (1000000); // 1MB/s over 10s
  Madara::Knowledge_Engine::Eval_Settings eval_settings;

  Madara::Knowledge_Record processes = knowledge.get ("processes.deployed");
  knowledge.set (".id", my_id);
  knowledge.read_file ("agent{.id}.view", "/image.jpg" eval_settings);
}
```

Advanced MADARA Features: Deadline Enforcement

In networks, especially in wireless networks, packet resends can result in long latencies between send and receive. In many real-time systems, old data is rarely useful and should be discarded in preference of new data.

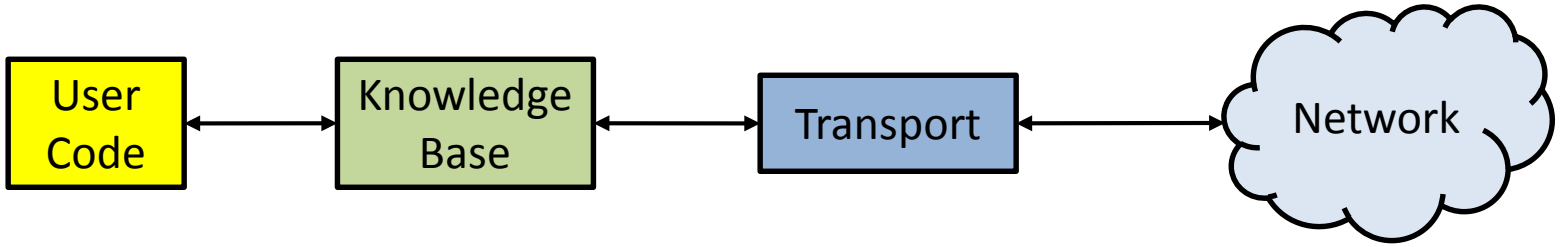


In MADARA, we facilitate both coarse-grained and fine-grained deadline enforcement.



We next discuss coarse-grained deadline enforcement.

Advanced MADARA Features: Deadline Enforcement



1. Setup normal transport settings
2. Set deadline to the number of acceptable seconds

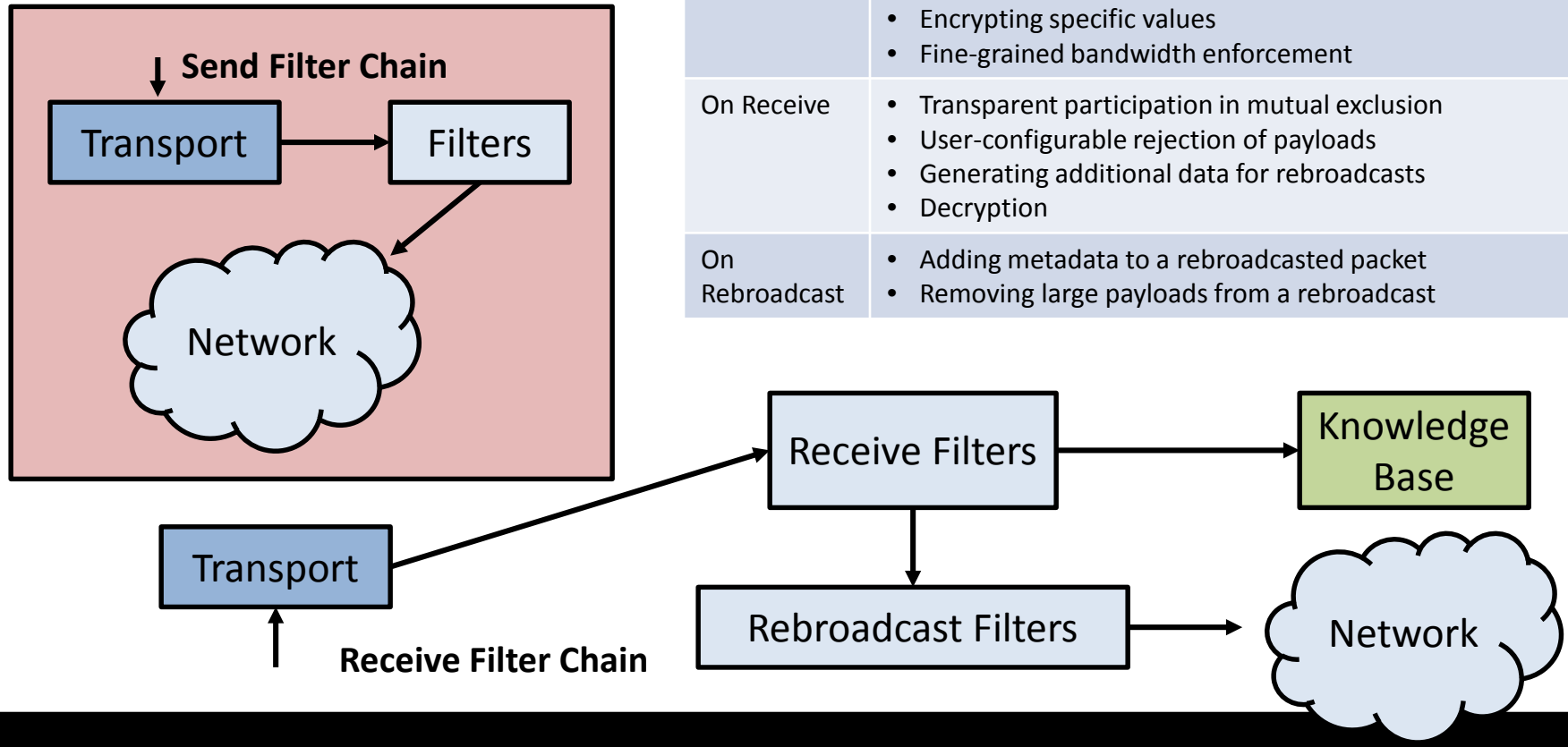
Example code

```
#include "madara/knowledge_engine/Knowledge_Base.h"

Madara::Knowledge_Record::Integer my_id (1);

int main (int argc, char ** argv)
{
  Madara::Transport::QoS_Transport_Settings settings;
  settings.hosts.push_back ("239.255.0.1:4150");
  settings.type = Madara::Transport::MULTICAST;

  settings.set_deadline (10);   // 5s deadline

  Madara::Knowledge_Engine::Knowledge_Base knowledge ("", settings);
  Madara::Knowledge_Engine::Eval_Settings eval_settings;

  Madara::Knowledge_Record processes = knowledge.get ("processes.deployed");
  knowledge.set (".id", my_id);
  knowledge.read_file ("agent{.id}.view", "/image.jpg" eval_settings);
  …
}
```

Advanced MADARA Features: Custom Filters

Custom filters in MADARA are native user functions that can be called by the transport layer.

Even	Example application use cases
On Send	• Converting a value to or from XML or plain text • Resizing, cropping, or shaping images or other payloads • Encrypting specific values • Fine-grained bandwidth enforcement
On Receive	• Transparent participation in mutual exclusion • User-configurable rejection of payloads • Generating additional data for rebroadcasts • Decryption
On Rebroadcast	• Adding metadata to a rebroadcasted packet • Removing large payloads from a rebroadcast



Advanced MADARA Features: Custom Filters: Arguments

Custom filters in MADARA are called with a specific vector of arguments that allow you to understand the context of invocation of the filter

Index	Description
[0]	Record being sent, received or rebroadcasted
[1]	Name of record
[2]	Operation type (SEND, RECEIVE, REBROADCAST)
[3]	Send bandwidth used in b/s over last 10s
[4]	Total bandwidth used in b/s over last 10s
[5]	Wall clock time of generation of message in seconds
[6]	Wall clock time of this operation in seconds
[7]	Domain (partition of knowledge updates)
[8]	Knowledge originator (source of the update)

Adding records for send/rebroadcast

The arguments vector is modifiable to provide developers with the ability to add metadata or new records that are generated during a send, receive, or rebroadcasted message

Index	Description
[n + 1]	Name of new record
[n + 2]	Value of new record
…	Repeat as needed with args.push_back function

Advanced MADARA Features: Custom Filters: Example: Encryption

We highlight the feature with an example of using Blowfish encryption with libtomcrypt on binary files.

1. Grab an unmanaged buffer of the file

2. Encrypt the file with a shared key

3. Set the return value to the encrypted file

4. Clean up the buffer and return the new result to be sent over the network

A simple Send Filter for encrypting each binary file payload

```
Madara::Knowledge_Record
encrypt (Madara::Knowledge_Engine::Function_Arguments & args,
         Madara::Knowledge_Engine::Variables & vars)
{
  Madara::Knowledge_Record result;
  if (args[0].is_binary_file_type ())
  {
    size_t size;
    unsigned char * buffer = args[0].to_unmanaged_buffer (size);

    blowfish_ecb_encrypt (buffer, buffer, &shared_key);

    result.set_file (buffer, size);

    delete buffer;
  }
  return result;
```

Advanced MADARA Features: Custom Filters: Example: Encryption

We highlight the feature with an example of using Blowfish encryption with libtomcrypt on binary files.

1. Grab an unmanaged buffer of the file

2. Decrypt the file with a shared key

3. Set the return value to the encrypted file

4. Clean up the buffer and return the new result, which will then be applied to the knowledge base

A simple Receive Filter for decrypting each binary file payload

```
Madara::Knowledge_Record
decrypt (Madara::Knowledge_Engine::Function_Arguments & args,
         Madara::Knowledge_Engine::Variables & vars)
{
  Madara::Knowledge_Record result;
  if (args[0].is_binary_file_type ())
  {
    size_t size;
    unsigned char * buffer = args[0].to_unmanaged_buffer (size);

    blowfish_ecb_decrypt (buffer, buffer, &shared_key);

    result.set_file (buffer, size);

    delete buffer;
  }
  return result;
```

Advanced MADARA Features: Custom Filters: Example: Encryption

We highlight the feature with an example of using Blowfish encryption with libtomcrypt on binary files.

1. Setup normal transport settings

2. Add the send and receive filters for all file types

3. Any files will be encrypted before they are sent and decrypted before being applied to knowledge base

A simple main function for adding the send and receive filters to a knowledge base

```
#include "madara/knowledge_engine/Knowledge_Base.h"

Madara::Knowledge_Record::Integer my_id (1);

int main (int argc, char ** argv)
{
Madara::Transport::QoS_Transport_Settings settings;
settings.hosts.push_back ("239.255.0.1:4150");
settings.type = Madara::Transport::MULTICAST;

settings.add_send_filter (Madara::Knowledge_Record::ALL_FILE_TYPES, encrypt);
settings.add_receive_filter (Madara::Knowledge_Record::ALL_FILE_TYPES, decrypt);

  Madara::Knowledge_Engine::Knowledge_Base knowledge ("", settings);
  Madara::Knowledge_Engine::Eval_Settings eval_settings;

  knowledge.set (".id", my_id);

knowledge.read_file ("agent{.id}.view", "/image.jpg" eval_settings);
}
```

Advanced MADARA Features: Custom Filters: Notes

- Filters can be added together to form a filter chain
- Filter chains are executed in the sequence they were added to the transport settings
- MADARA provides a set of generic filters in madara/filters/Generic_Filters.h

Name	Description
discard	Discard all records
discard_nonprimitives	Discard all non-primitive types
discard_nonfiles	Discard all non-file types
log_args	Prints all arguments to the MADARA logger (default stderr). This filter is very useful for debugging filters and applications.

Upcoming Features (Short term--within 2 months)

- **Visualization System (DSML)**
  - Aids in designing new MADARA applications

- **Update Aggregation Filters**
  - Informs developer of complete context of aggregate update for filtering purposes
  - Unlike Update Filters, does not provide args vector
  - Provides a STL map of variable names to records
  - Provides a Transport Context

More Information About MADARA

Website: madara.googlecode.com
Wiki | Library Documentation | Installation | Developer Blog

SMASH project: smash-cmu.googlecode.com | Youtube Demo

Main Developers: James Edmondson, James Root (Java port)

Special thanks: Sebastian Echeverria, Anton Dukeman, Subhav Pradhan, Ben Bradshaw, Anthony Rowe, Luis Pinto, and the AMS group at SEI

The results of the MADARA project could not have been possible without support and feedback from various colleagues, students, and coworkers, and collaboration with researchers in funded projects from the universities and organizations below. **THANK YOU!**

VANDERBILT UNIVERSITY

Carnegie Mellon University

Software Engineering Institute
Carnegie Mellon