Building a real-time Grid protocol analyser

UNIVERSITY
of
GLASGOW

Jonathan Paisley
Department of Computing Science

The Grid

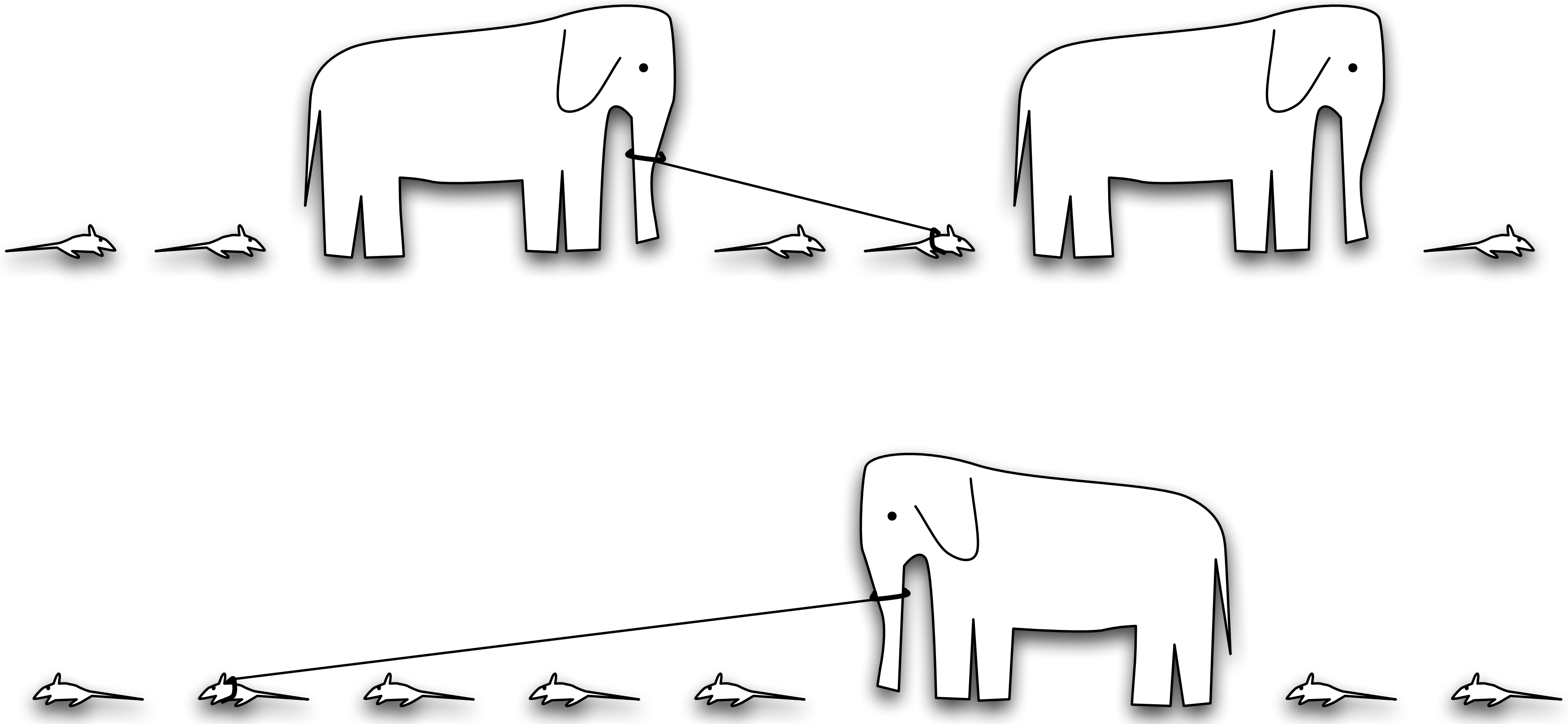
Wide-area distributed computing

Lots of funding

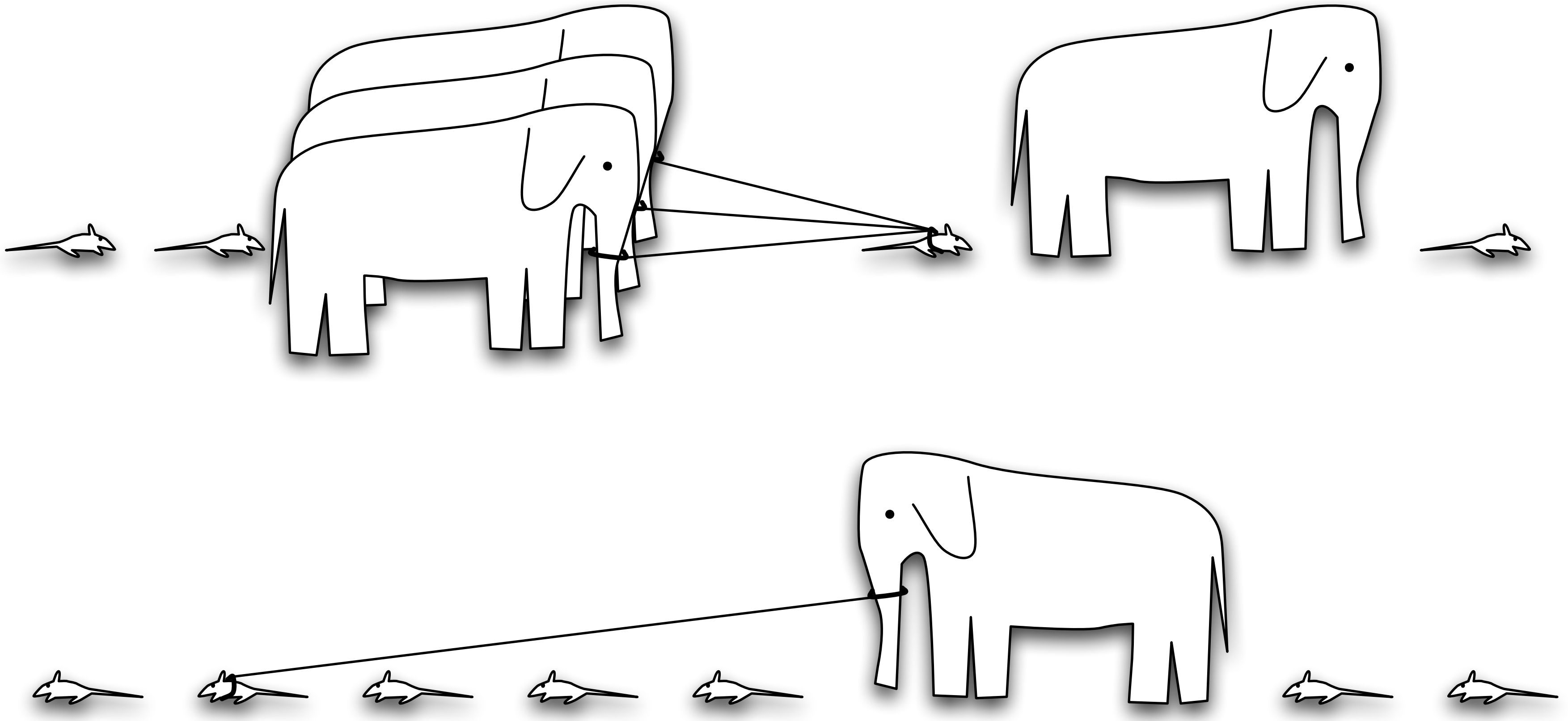
Network operators need to support it

Traffic dominated by bulk data transfer

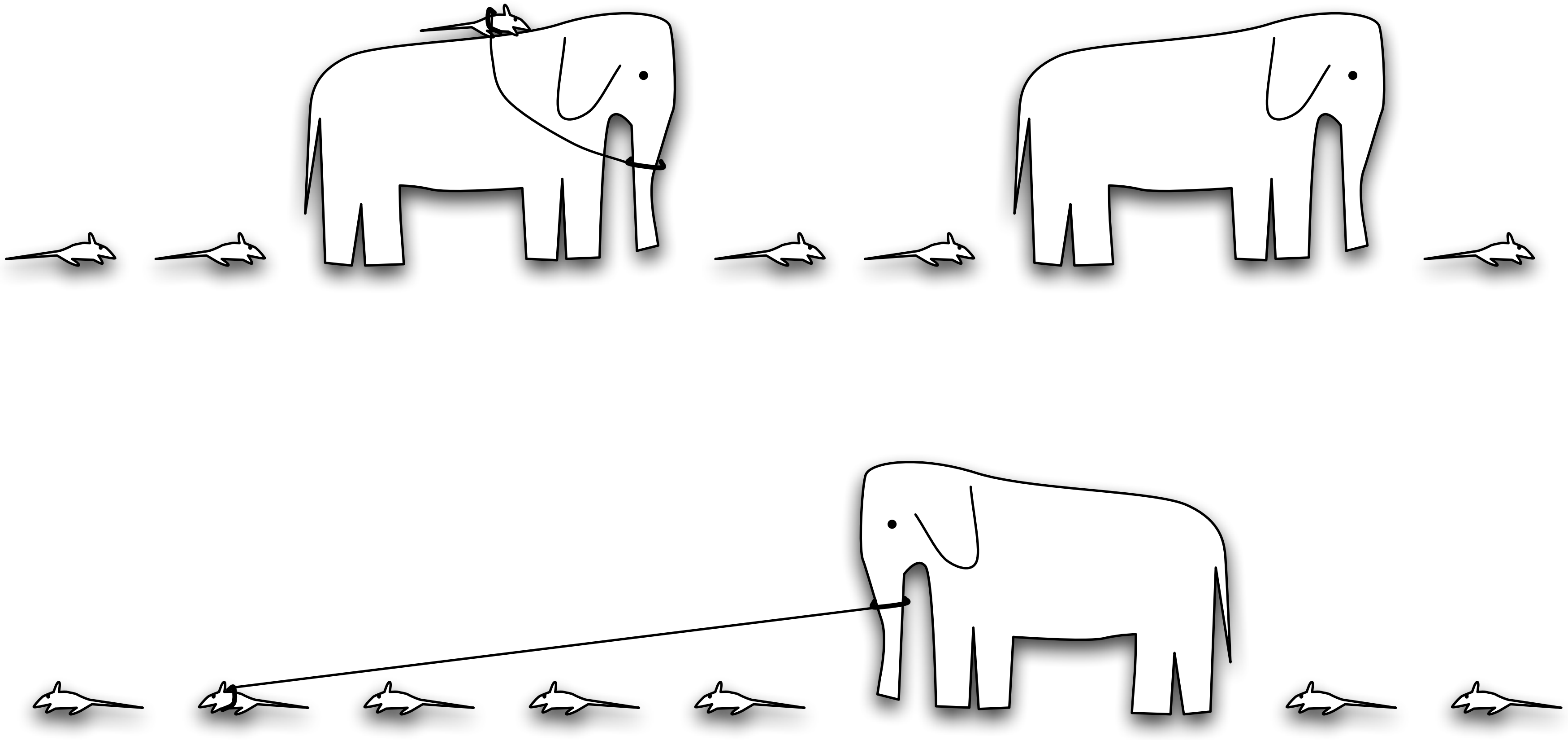
Elephants and Mice



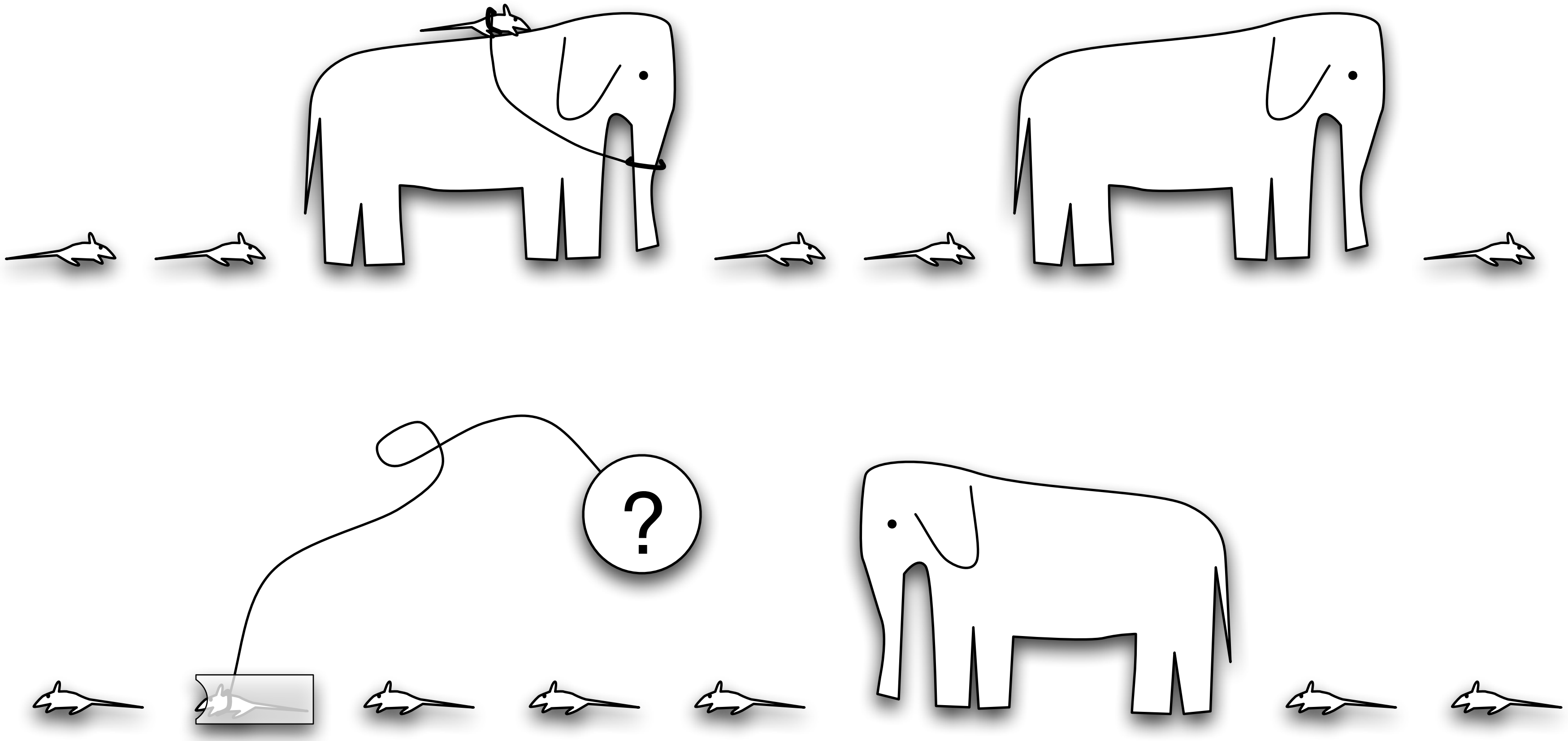
Multiple Elephants and Mice



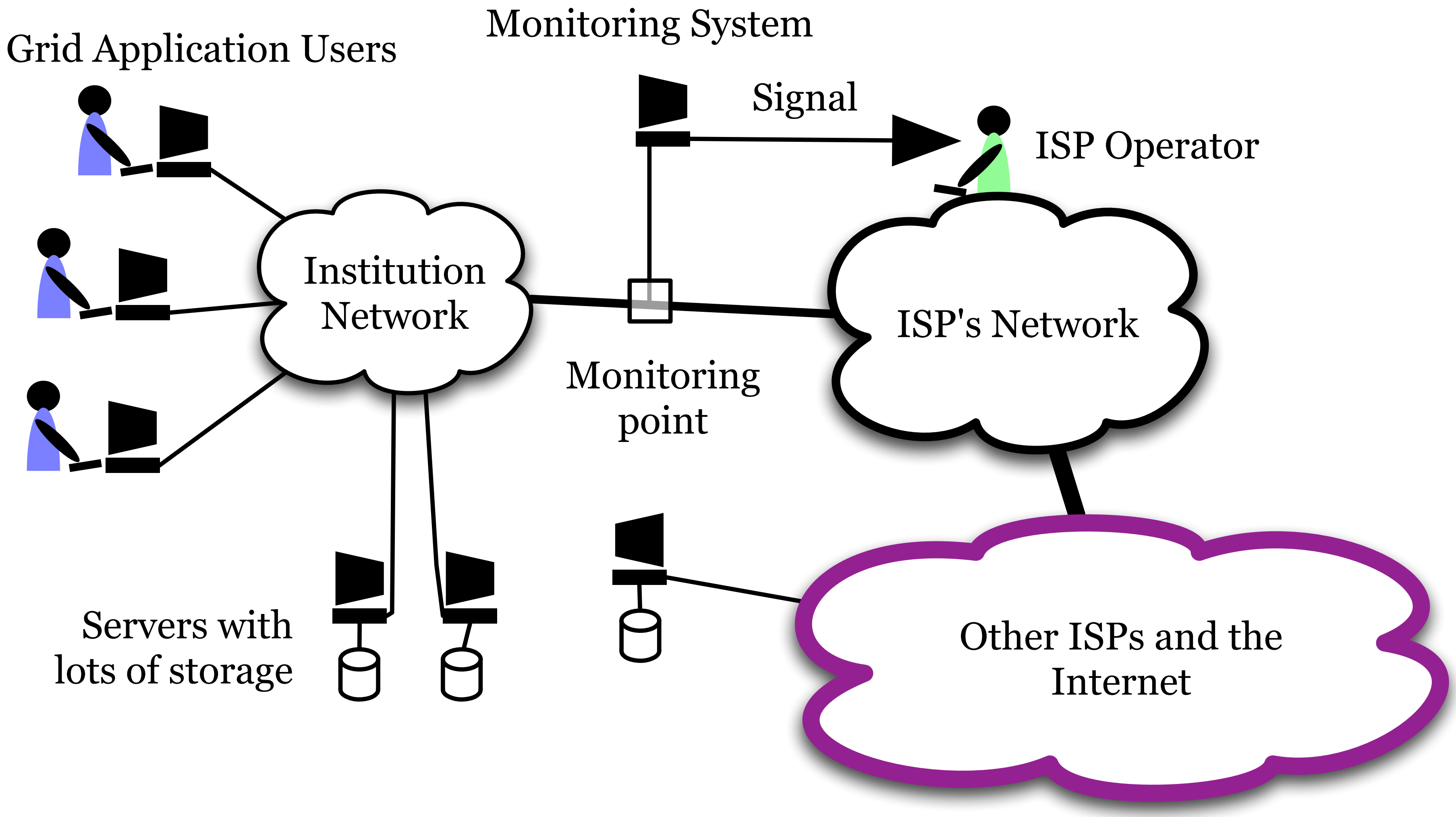
Elephants and Mixed Mice



Elephants and Cipher Mice



Grid Monitoring



Approach

Interpret protocol to learn about associated bulk connections

Report on transfer sizes

Be able to deal with mixed control-data flows

DAG-based Network Monitor

Just a PC with special network monitoring card.
Example: 2.8 GHz dual Xeon, 2+ GB memory

Image Source: Endace Measurement Systems

Similarity to NIDS

NIDS = Network Intrusion Detection System

For example: Bro

Does protocol analysis (FTP, SMTP, etc)

Needs port-based filter

Full reassembly of every monitored flow

Too slow

Design Goals

Leverage DAG ring buffer architecture

Capable of processing at GigE line rate

Support full cleartext protocol analysis

Efficiently handle mixed control/data

Assumptions and Principles

TCP only

Applications under study not used maliciously

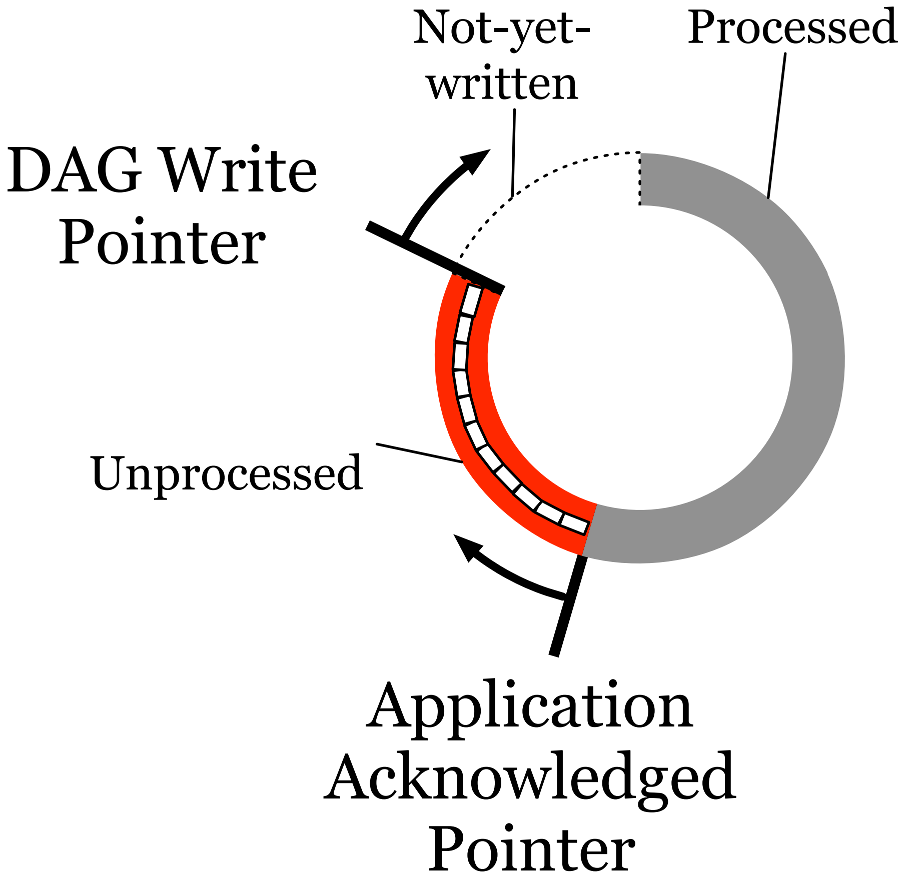
Minimise memory copies

Minimise heap allocation

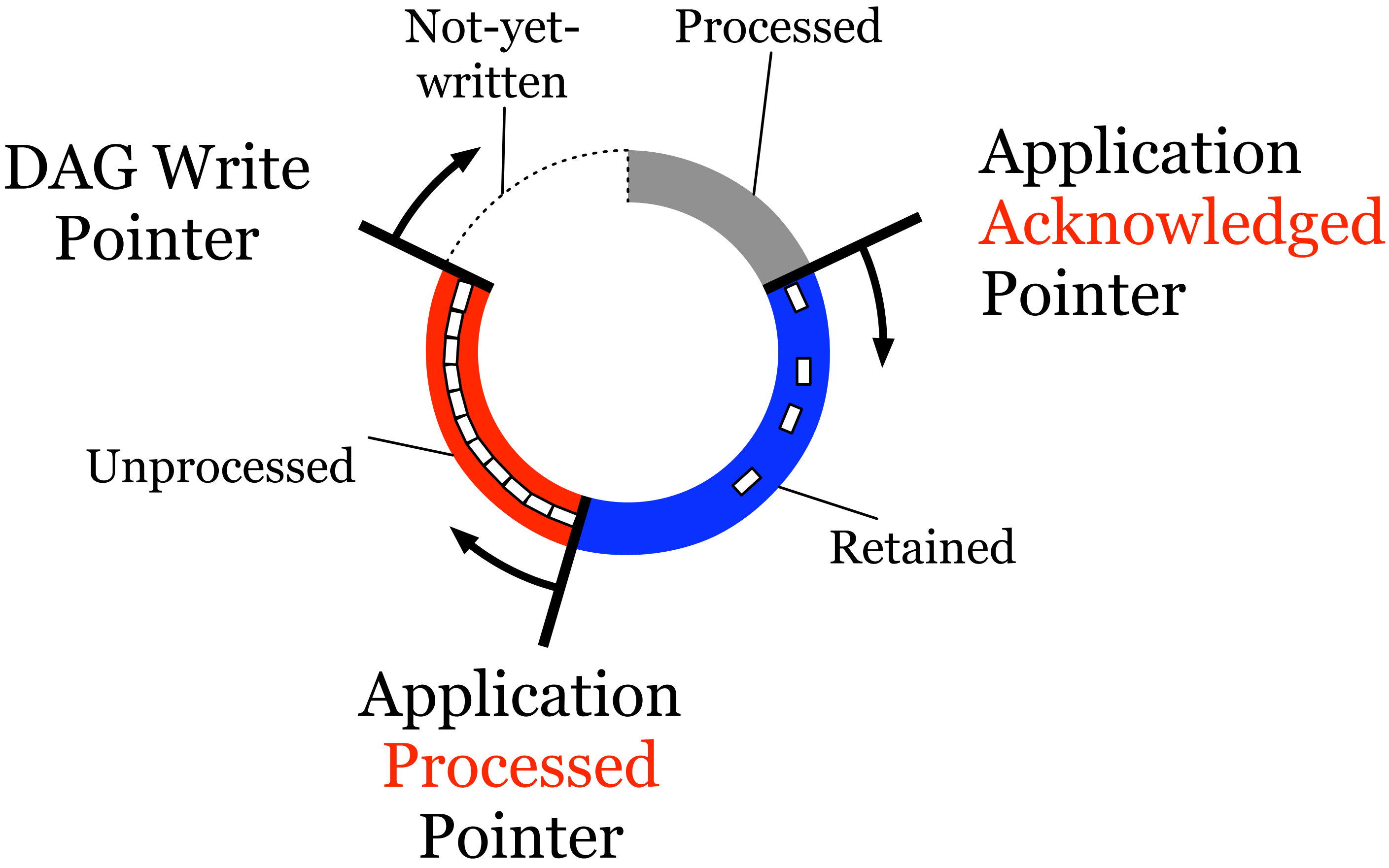
Process packets as soon as possible

Single-threaded, data-driven

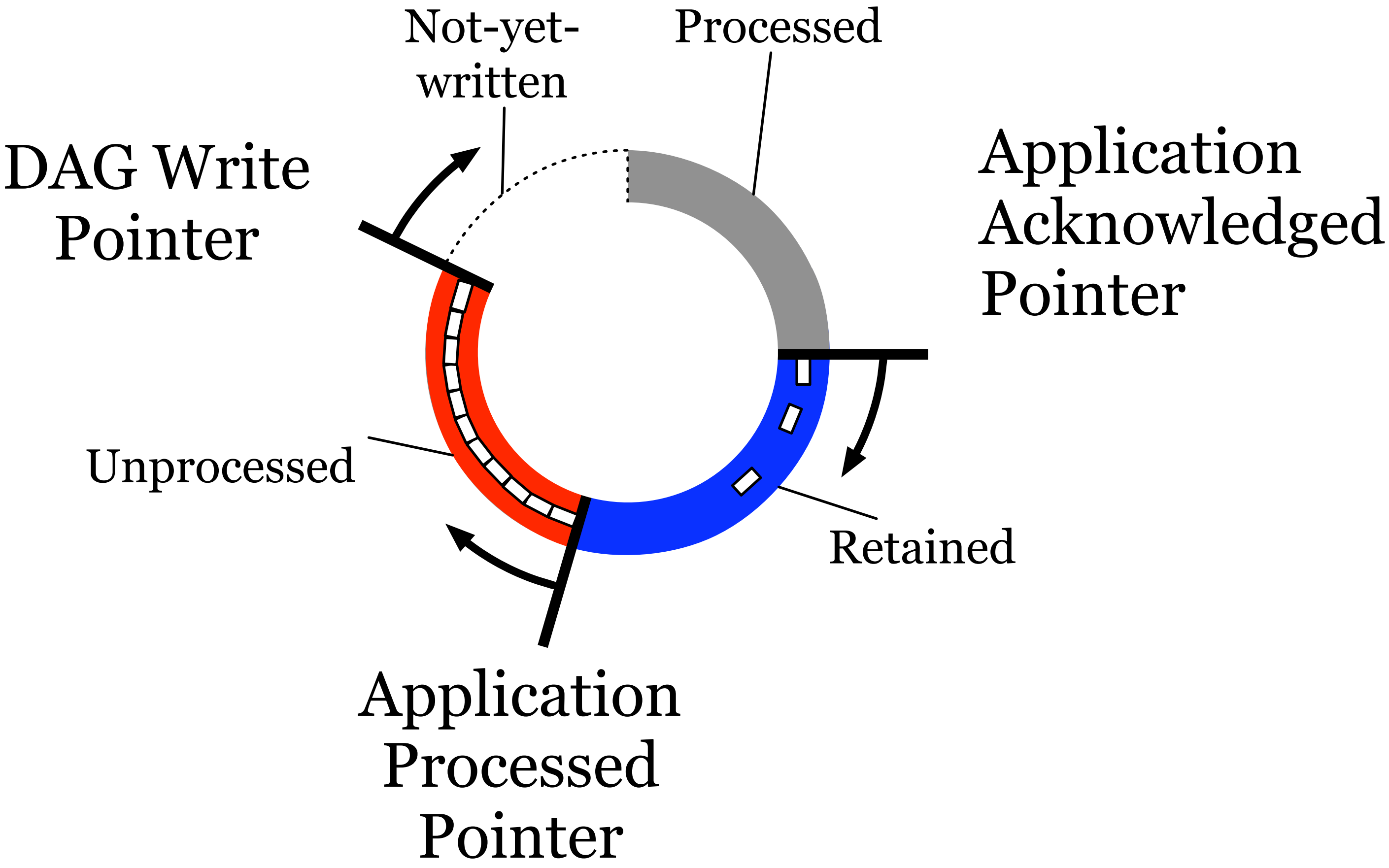
DAG Ring Buffer



DAG Ring Buffer



DAG Ring Buffer


Not-yet-written
Processed
DAG Write Pointer
Application Acknowledged Pointer
Unprocessed
Retained
Application Processed Pointer

Writing Protocol Analysers

Passive monitor sees both flow directions

Code to track state -> generate events

State machines can be complex

Threaded programming style is easier

… but runtime cost normally higher

ProtoThreads

Similar to co-routines/continuations

Implemented using a C switch statement

State maintained in a structure

Context switching by stack unwinding

Analyser Example

```
void AnalyserClass::AnalyserMain()
{
    // Read function id and num args
    READ(OrigFlow, 2);
    func_id = *(uint16_t*)data;
    READ(OrigFlow, 2);
    num_args = *(uint16_t*)data;

    for (i=0;i<num_args;i++) {
      READ(OrigFlow, 4);
      len = *(uint32_t*)data;
      // Read the argument, but we
      // only need the first 200 bytes
      READ_AND_SKIP(OrigFlow, len, 200);
      // ... process the argument
    }
    READ(RespFlow, 4);
    result_value = *(uint32_t*)data;
}
```

Analyser Example

```
void AnalyserClass::AnalyserMain()
{
    // Read function id and num args
    READ(OrigFlow, 2);
    func_id = *(uint16_t*)data;
    READ(OrigFlow, 2);
    num_args = *(uint16_t*)data;

    for (i=0;i<num_args;i++) {
      READ(OrigFlow, 4);
      len = *(uint32_t*)data;
      // Read the argument, but we
      // only need the first 200 bytes
      READ_AND_SKIP(OrigFlow, len, 200);
      // ... process the argument
    }
    READ(RespFlow, 4);
    result_value = *(uint32_t*)data;
}
```

May block here!

Scalability

Presently limited to single processor

Auxiliary flow tracing complicated by concurrent processing

Could use retained packet scheme for all flows:
    gives extra 1-2 seconds buffering

Evaluation

Informal testing carried out during development

Negligible load for ~900Mbps mixed control/ data SRB flow

Initial testing with larger number of connections with flat-out* replay of IP header traces ~10-15% load

* 250Mbps, 5000 new connections per second.

Encrypted Analysis Ideas

What can we know about encrypted traffic?

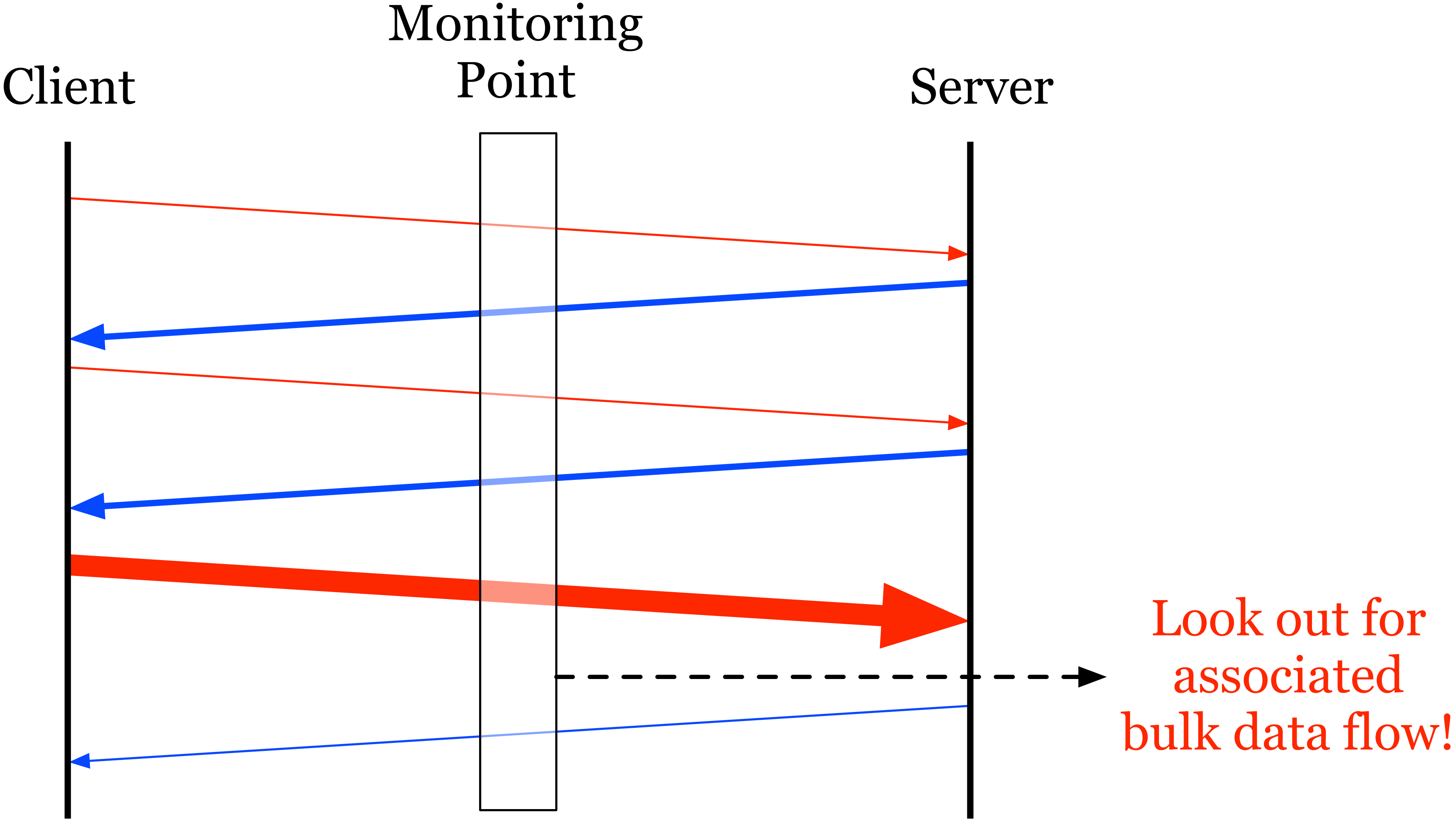
- Messages: (direction, size*, timing)
- Lack of messages (timeouts)

If we understand framing protocol:

- can get application-level messages

* with some bounded error

Requests and Responses



Look out for associated bulk data flow!

Approaches

Hidden Markov Models?

Naïve Bayesian Classifier?

Other work:

- SSH password typing analysis
- HTTPS request analysis by URL lengths
- Sideband attacks on encryption algorithms

Summary

Built (hopefully) fast system for real-time protocol analysis work. Evaluation pending.

Support for efficient handling of mixed control/data protocols.

Coding of protocol analysers simplified by rich lightweight threaded interface.

Starting work on classifying and event reporting of encrypted traffic.