EECS 262a
Advanced Topics in Computer Systems
Lecture 15

Agile VM Migration/
Global Data Plane/DataCapsules
October 12th, 2022

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Papers

You Can Teach Elephants to Dance
Kiryong Ha, Yoshihisa Abe, Thomas Eiszler, Zhuo Chen, Wenlu Hu, Brandon Amos, Rohit Upadhyaya, Padmanabhan Pillai, Mahadev Satyanarayanan, 2017

Global Data Plane: A Federated Vision for Secure Data in Edge Computing
Nitesh Mor, Richard Pratt, Eric Allman, Ken Lutz, and John Kubiatowicz.

Thoughts?

Recall: Edge Computing

Wikipedia:

Edge computing is a distributed computing paradigm in which computation is largely or completely performed on distributed device node known as smart devices or edge devices as opposed to primarily taking place in a centralized cloud environment.

Why compute on the edge?
- Latency: Importance of human interactivity
- Privacy: Keep sensitive data local
- Reliability: Keep computing during network partitions

Why VM Encapsulation at Edge?

ightweight mechanisms give performance and scalability
- Docker, Linux containers, other in-kernel mechanisms
- Focus on lightweight encapsulation

ther attributes to consider (that favor VMs):
- *Safety*: Protect infrastructure from potentially malicious software
- *Isolation*: Hide actions of mutually untrusting executions from one another on multi-tenant cloudlet
- *Transparency*: Ability to run unmodified app code without recompiling or relinking. Allow reuse of existing software for rapid deployment
- *Deployability*: The ability to easily maintain cloudlets in the field and create mobile apps that have high likelihood of finding software-compatible cloudlet anywhere in the world.

short – VMs have many advantages in the edge environment
- Can you get above advantages without full guest OS?

Applications explored

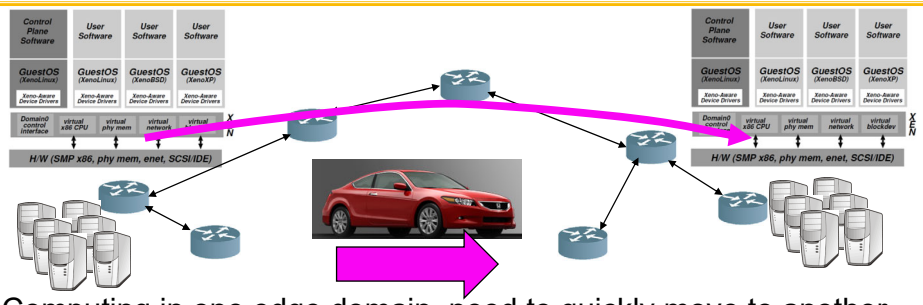
.pp1: periodically sends accelerometer readings from a mobile device to a inux back-end that performs a compute-intensive physics simulation, and eturns an image to be rendered

.pp2: ships an image from a mobile device to a Windows back-end, where ace recognition is performed and labels corresponding to identified faces ar eturned

.pp3: an augmented reality application that ships an image from a mobile evice to a Windows back-end that identifies landmarks, and returns a nodified image with these annotations

.pp4: ships an image from a mobile device to a Linux back-end that perform bject detection, and returns the labels of identified objects

Agile VM Handoff for Edge Computing



- Computing in one edge domain, need to quickly move to another when operating conditions change
  - Mobile devices (cars, people, etc) about to go out of range
  - Server hardware overloaded (/oversubscribed) or about to fail
- Contrast with Live Migration (Last time)
  - More than just downtime of service: total completion time matters
  - WAN links between domains means that lower BW available
  - Importance of cutting all ties to source hardware quickly

Basic Design and Implementation

- VM Handoff principles:
  - *Every non-transfer is a win:* Use deduplication, compression and delta-encoding to eliminate transfers
  - *Keep the network busy:* Network bandwidth is a precious resources, and should be kept at the highest level of utilization
  - *Go with the flow:* Adapt at fine time granularity to network bandwidth and cloudlet compute resources
- Contrast with Live Migration:
  - Live Migration tries to balance impact on network resources
  - Live Migration tries to minimize downtime
  - Live Migration assumes source hardware can stay up longer
  - Live Migration assumes destination has no common "base" from which to start transfer process (and enable deduplication from base)
- Note that Data Deduplication is an essential element of WAN optimization appliances
  - Extremely successful at reducing network transmission
  - Done on transmitted data ("read-only") thus does not affect consistency

Poor Agility of Live Migration

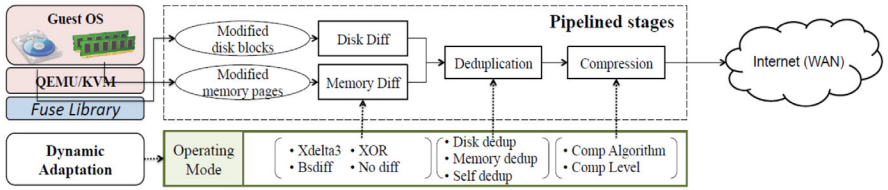
VM	Total time	Down time	Transfer Size
App3	3126s (39%)	7.63s (11%)	3.45 GB (39%)
App4	726s (1%)	1.54s (20%)	0.80 GB (1%)

Average and Relative standard deviation of 3 runs. These results are extracted as a preview of Figure 7 in Section 5.3 See Figure 7 for full details.

Figure 2: Total Completion Time of Live Migration (10 Mbps)

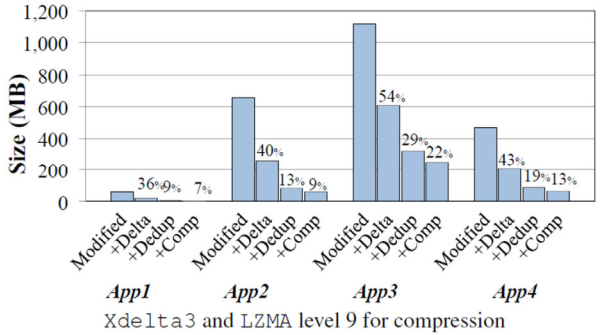
- Total Time field (for leaving source machine) *VERY* large
  - Could another approach for movement bring down Total Time without increasing Down Time too much?
- Total amount of data Transferred also large!
  - In WAN environment, dealing with lower BW (10 Mbps or lower)
  - Ways of bringing this down?

The compression pipeline for this paper



- Start with base images
  - Assumption is that these are distributed to endpoints in advance
  - How many? A "handful of base images"
- Track modified blocks
  - Disk block changes tracked by clever use of FUSE interface
  - Memory changes tracked by virtual machine
- Use various differencing algorithms
- Apply deduplication algorithms
  - Compare hashes of blocks in base images and blocks already sent
- Compress results (using various algorithms and intensities)

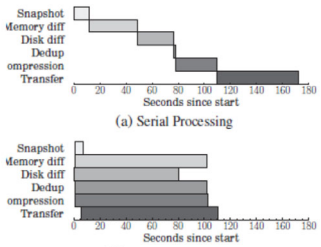
Reduction in size of transmitted data



Xdelta3 and LZMA level 9 for compression

- Degree of Compression
  - Delta+Deduplication+Compression
  - Makes significant reduction over modified blocks (1/5 to 1/10 original size)
- Tradeoff of computation for bandwidth
  ⇒ Dynamic adaptation as operating point and/or BW change

Pipelined Execution



App4: xdelta3 and LZMA level 5

Figure 5: Serial versus Pipelined Data Reduction

- Pipeline operations to start filling network quickly
  - These operations are naturally pipelined at block-level (4K)
  - Also naturally parallelized with multicore
- For this example, pipelining saves 36% completion time

Results under different conditions:

BW	5 Mbps		10 Mbps		15 Mbps		20 Mbps		25 Mbps	
Time (s)	Total	Down	Total	Down	Total	Down	Total	Down	Total	Down
App 1	25.1	4.0 (5%)	24.6	3.2 (29%)	23.9	2.9 (38%)	23.9	3.0 (38%)	24.0	2.9 (43%)
App 2	247.0	24.3 (3%)	87.4	15.1 (10%)	60.3	11.4 (8%)	46.9	7.0 (14%)	39.3	5.7 (25%)
App 3	494.4	24.0 (4%)	257.9	13.7 (25%)	178.2	8.8 (19%)	142.1	7.1 (24%)	121.4	7.8 (22%)
App 4	113.9	15.8 (6%)	66.9	7.3 (42%)	52.8	5.3 (12%)	49.1	6.9 (12%)	45.0	7.1 (30%)

(a) 1 CPU core

BW	5 Mbps		10 Mbps		15 Mbps		20 Mbps		25 Mbps	
Time (s)	Total	Down	Total	Down	Total	Down	Total	Down	Total	Down
App 1	17.3	4.1 (6%)	15.7	2.5 (4%)	15.6	2.2 (14%)	15.4	2.0 (19%)	15.2	1.9 (20%)
App 2	245.5	26.5 (7%)	77.4	14.7 (24%)	48.5	6.7 (15%)	36.1	3.6 (12%)	31.3	4.1 (17%)
App 3	493.4	24.5 (10%)	250.8	12.6 (13%)	170.4	9.0 (17%)	132.3	7.3 (20%)	109.8	6.5 (22%)
App 4	111.6	17.2 (7%)	58.6	5.5 (5%)	43.6	5.5 (31%)	34.1	2.1 (22%)	30.2	2.1 (26%)

(b) 2 CPU cores

- Observations:
  - In general, Total Time decreases with multi-core and/or
  - Downtime mostly decreases with multi-core
- At low BW:
  - slow transfers allow source VM to dirty more pages during transfer process, increasing transfer volume
  - but also leaving more time to spend getting better compression

Comparison with other options

VM	Approach	Total time (s)	Down time (s)	Transfer size (MB)
App1	VM handoff	16 (3 %)	2.5 (4%)	7
	Live Migration	229 (1 %)	2.4 (22 %)	235
App2	VM handoff	77 (4 %)	14.7 (24 %)	84
	Live Migration	6243 (68 %)	5.5 (17 %)	6899
App3	VM handoff	251 (1 %)	12.6 (13 %)	276
	Live Migration	3126 (39 %)	7.6 (11 %)	3533
App4	VM handoff	59 (1 %)	5.5 (5 %)	61
	Live Migration	726 (1 %)	1.5 (20 %)	82

Figure 7: VM handoff vs. KVM/QEMU Live Migration at 10 Mbps

VM	Approach	Total time	Down time	Transfer size
App1	VM handoff	15.7s	2.5s	7.0 MB
	Docker	6.9s	6.9s	6.5 MB
App4	VM handoff	58.6s	5.5s	61 MB
	Docker	118s	118s	98 MB

Only App1 and App4 are shown because `Docker-CRIU` works for only Linux Apps.

Figure 8: VM handoff vs. Docker at 10 Mbps

- Total time is *MUCH* shorter with VM Handoff vs Live Migration (by more than order of magnitude)
- Down time increases with VM Handoff, but by < factor of 3
- Docker migration requires separate checkpoint operation in order to move, thereby increasing expense considerably!

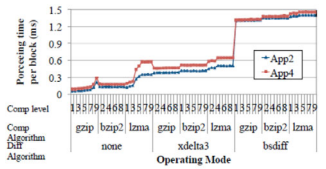
Series of optimizations

- Automatic Adaptation Heuristic
  - Adjust processing WRT network BW
  - Measure throughput (bytes/sec) for processing one standard block
  - Contrast with throughput to send over network (also bytes/sec)
  - Maximize $Thru_{system}$ against possible system modes using offline model
- Adaptation using "observation"
  - Although processing cost and network BW depends on application, the *relative* behavior from one mode to next is "similar"
- Other Optimizations
  - Randomized transfer order to avoid bustiness from spatial locality of page mods
  - Iterative transfer (similar to live migration) using thresholds of modified state limited by time to put limit on total transfer time

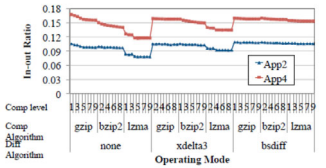
$$Thru_{system} = min(Thru_{processing}, Thru_{network}) \quad (5)$$

$$Thru_{processing} = \frac{1}{\sum_{1 \le i \le n} P_i}$$

$$Thru_{network} = \frac{\text{Network Bandwidth}}{(R_1 \times \cdots \times R_n)} \quad (6)$$



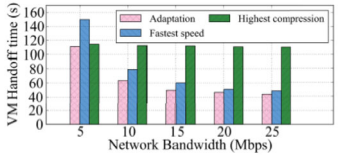
(a) Processing time (P) in different operating modes



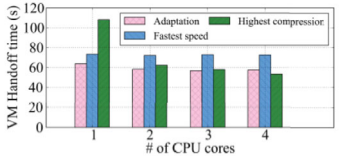
(b) Out-in ratio (R) in different operating modes

(G: `Gzip`, B: `Bzip2`, L: `LZMA`)

Adaptation results



(a) Varying Network Bandwidth (1 CPU core)



(b) Varying CPU Cores (10 Mbps bandwidth)

Figure 15: Adaptation vs Static Modes (App4)

BW (mbps)	Approach	Total time	Down time
5	Adaptation	113.9s (3 %)	15.8s (6 %)
	Best static	111.5s (1 %)	15.9s (12 %)
	Top 10%	128.3s (2 %)	20.7s (9 %)
10	Adaptation	66.9s (6 %)	7.3s (42 %)
	Best static	62.0s (1 %)	5.0s (11 %)
	Top 10%	72.1s (1 %)	4.8s (3 %)
20	Adaptation	49.1s (8 %)	6.9s (12 %)
	Best static	45.5s (3 %)	8.1s (15 %)
	Top 10%	48.5s (1 %)	4.9s (11 %)
30	Adaptation	37.0s (4 %)	2.6s (47 %)
	Best static	34.3s (2 %)	2.1s (8 %)
	Top 10%	48.5s (1 %)	4.8s (3 %)

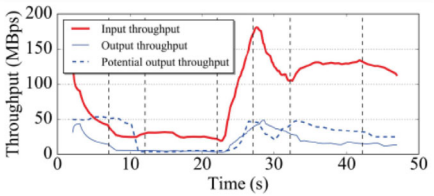
Relative standard deviation in parentheses

Figure 16: Adaptation vs Static Mode for App4 (1 core)

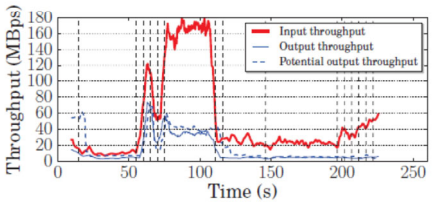
- Adjusts to network bandwidth variations and parallelism
- Compares well with top-10 lowest times

Adaptation to changing conditions

- Application 4 (Figure 18a)
  - First dashed line: chooses high processing, high compression settings (LZMA, level 9) to deal with low network BW
  - At 20s, switches to lightest weight mode (GZIP, level 1, no differencing)
- Application 3 (Figure 18b)
  - First point, high processing, high compression (LZMA, level 9)
  - At 58s, major decision make to switch back to GZIP compression to avoid being processing bound
  - At 100s+small amount, switches back to LZMA, level 9



(a) App4: 5 Mbps → 35 Mbps at 20 s



(b) App3: 5 Mbps → 35 Mbps at 20 s → 5 Mbps at 100 s

Figure 18: Adaptation for Network Bandwidth

Support for legacy apps through VNC

		Total time (s)	Downtime (s)
Live Migration	1 Gbps	946.4 (0 %)	0.2 (52%)
VM handoff	1 Gbps	87.8 (24 %)	6.6 (20 %)
	100 Mbps	111.0 (29 %)	7.4 (20 %)
	25 Mbps	101.8 (3 %)	10.8 (4 %)
	20 Mbps	114.6 (2 %)	13.2 (6 %)
	15 Mbps	584.8 (7 %)	16.2 (5 %)
	10 Mbps	656.4 (1 %)	21.4 (10 %)

Average of 5 runs, with RSD in parentheses.
Windows 7 guest OS, 32 GB disk image, 1 GB memory
Added latency: 0 for 1 Gbps; 25 ms for 100 Mbps; 50 ms for all other bandwidths

Figure 19: VM handoff for WAN VDI

- Assume mobile user runs legacy apps on Edge Resources using a virtual desktop
  - Client is "thin", namely just putting bits on screen
- Possible switchover point might involve things like getting on/off aircraft or other major transitions
  - So – perhaps 6-21 seconds of downtime not a big deal?

Summary

'M encapsulation at the edge
- For safety, isolation, transparency, deployability

)ptimize for total migration time
- Important aspect not necessarily downtime

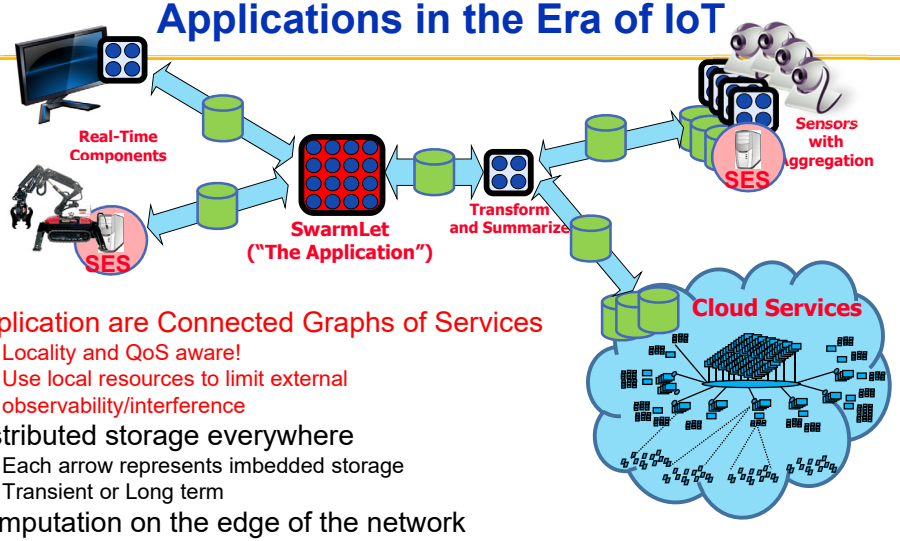
.ggressive use of data deduplication and adaptive compression
- Assume that a variety of "base" images available to use for eliminating unnecessary traffic
- Very similar to WAN optimization devices

Is this a good paper?

What were the authors' goals?
What about the evaluation/metrics?
)id they convince you that this was a good system/approach?
Were there any red-flags?
What mistakes did they make?
)oes the system/approach meet the "Test of Time" challenge?
low would you review this paper today?

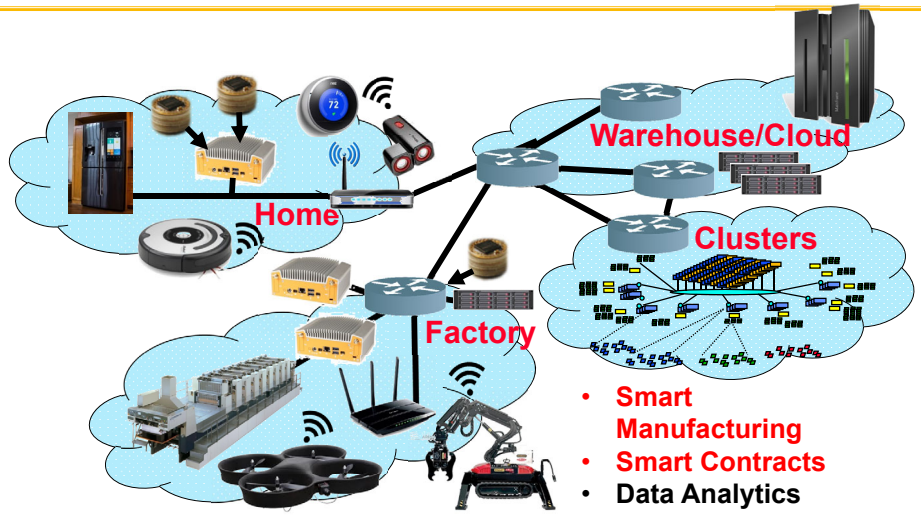
BREAK

Applications in the Era of IoT



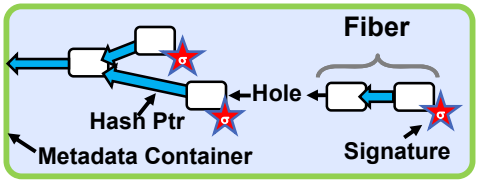
- Application are Connected Graphs of Services
  - Locality and QoS aware!
  - Use local resources to limit external
  - observability/interference
- Distributed storage everywhere
  - Each arrow represents imbedded storage
  - Transient or Long term
- Computation on the edge of the network
  - Perhaps Secure Enclaves (SES) for trusted computation…?
  - Rapid launching of computation to close resources

A Physical View of these Applications: Distributed, Ad Hoc, and Vulnerable



- **Smart Manufacturing**
- **Smart Contracts**
- **Data Analytics**
- **Machine Learning**

The Data-Centric Vision: Cryptographically Hardened Data Containers



- Inspiration: Shipping Containers
  - Invented in 1956. Changed everything!
  - Ships, trains, trucks, cranes handle *standardized format containers*
  - *Each container has a unique ID*
  - *Can ship (and store) anything*
- *Can we use this idea to help?*

- DataCapsule (DC):
  - Standardized metadata wrapped around opaque data transactions
  - Uniquely named and globally findable
  - Every transaction explicitly sequenced in a hash-chain history
  - Provenance enforced through signatures
- Underlying infrastructure assists and improves performance
  - Anyone can verify validity, membership, and sequencing of transactions (like blockchain)

Why does this help?

The "Networking" effect (Pun Intended!)

- Standardization ⇒ Infrastructure proliferation that benefits everyone
- Federation ⇒ Enable a market of service providers

Data becomes a first-class entity in the network!

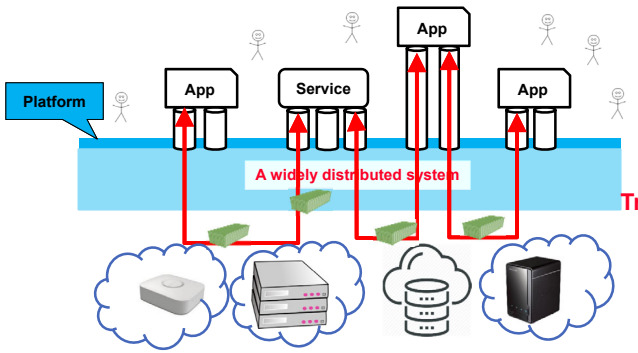
- Asserts its own requirements for security, privacy, which are enforced via cryptography
- Independent of physical location – policies can target durability, QoS, availability, etc
- No application silos – data producers own and chose how to share their information
- Network is informed about the information that it is carrying and where it may go

First (Necessary) Step:
Network Cannot Enforce what is not Specified!

Related information bundled and kept together as it migrates

- Provenance and data ordering part of all information usage
- Information labeled with meta-data about (1) Where it is allowed to be within the network, and (2) Who is allowed to view and interact with it, (3) Who is allowed to modify it.

A Platform Approach: the Utility-Provider Model [Ships, Trains, Trucks, and Cranes]

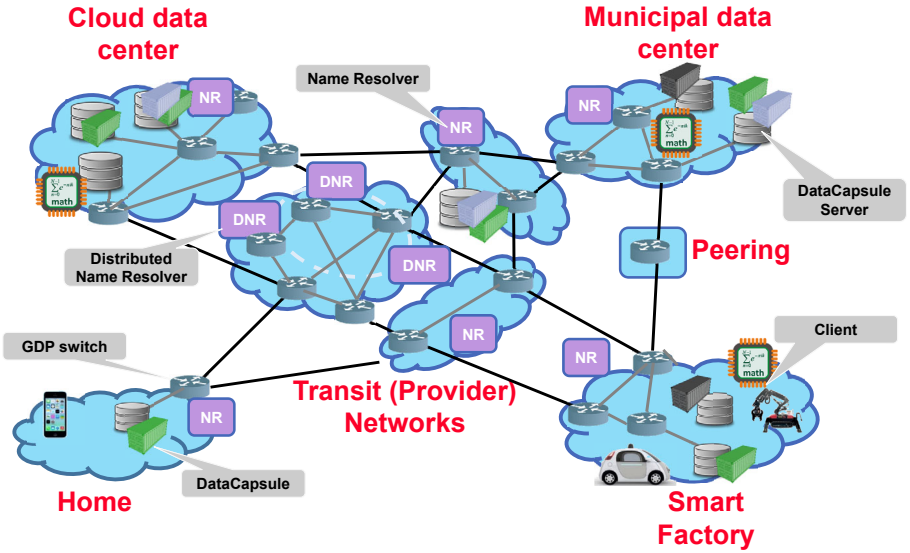


Platform Users: Apps/Services

Global Data Plane: Routing, Multicast, Trust Domains, Accounting

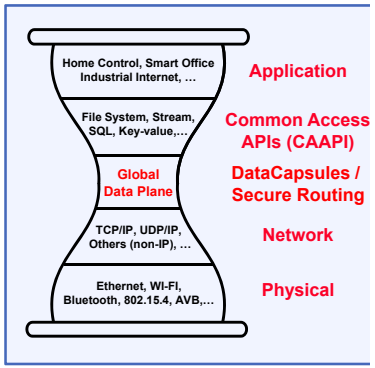
Utility Providers: Heterogeneous Infrastructure Storage, Transport, QoS

A Physical View of the GDP

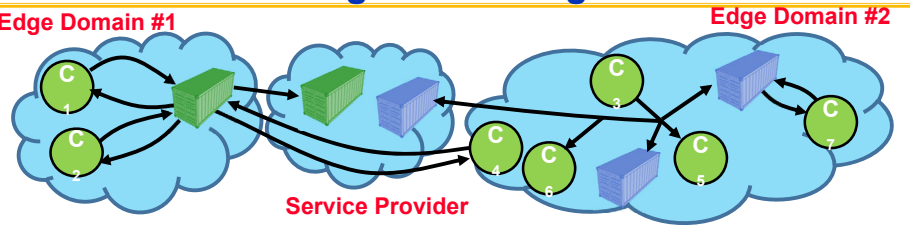


Refactoring of Applications around Security, Integrity, and Provenance

- Goal: A thin Standardized entity that can be easily adopted and have immediate impact
  - Can be embedded in edge environments
  - Can be exploited in the cloud
  - Natural adjunct to Secure Enclaves for computation
- "Eye-Of-The-Needle" proposition:
  - Thin enough that it will be adopted and enhanced by the most people
  - Powerful enough that application writers can do whatever they need to do
- DataCapsules ⇒ bottom-half of a blockchain?
  - Or a GIT-style version history
  - Simplest mode: a secure log of information
  - Universal unique name ⇒ permanent reference
- Applications writers think in terms of traditional storage access patterns:
  - File Systems, Data Bases, Key-Value stores
  - Called Common Access APIs (CAAPIs)
  - DataCapsules are always the *Ground Truth*



Global Data Plane (GDP) and the Secure Datagram Routing Protocol



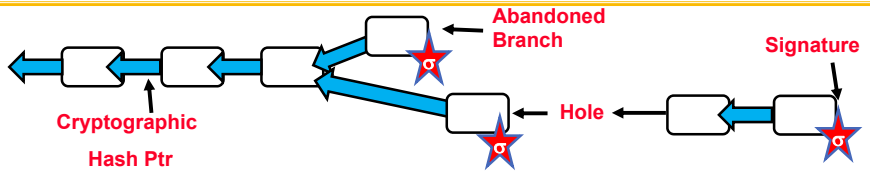
Flat Address Space Routing
- Route queries to DCs by names, independent of location (e.g. no IP)
- DCs move, network deals with it
- Short-term Channels ("μ-SSL channels")

Black Hole Elimination: Delegation of Names
- Only servers authorized by owner of DC may advertise DC service

Routing only through domains you trust!
- Secure Delegated Flat Address Routing

- Secure Multicast Protocol
  - Only clients/DC storage servers with proper (delegation) certificates may join
- Queries (messages) are *Fibers*
  - Self-verifying chunks of DataCapsules
  - Writes include appropriate credentials
  - Reads include proofs of membership
- Incremental deployment as an overlay
  - Prototype tunneling protocol ("GDPinUDP")
  - Federated infrastructure w/routing certificates

Use of Single-Writer Semantics

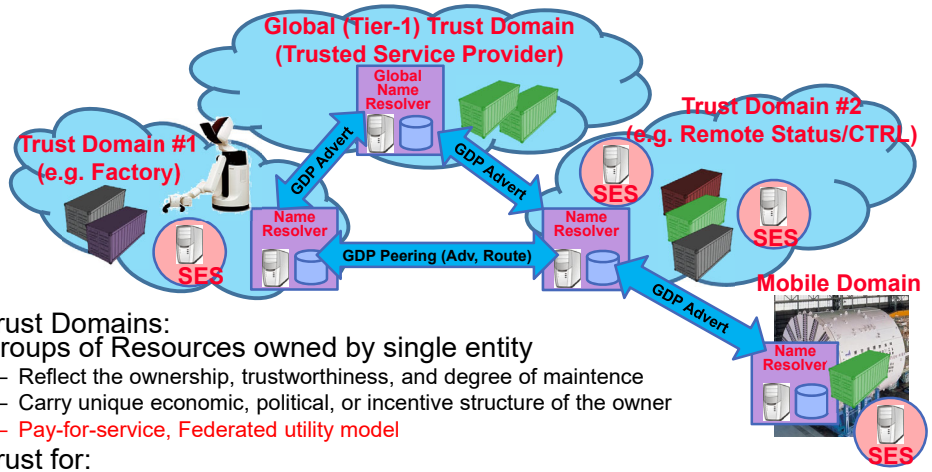


- Tradeoff between power of writer (or proxy) and degree of branching and/or holes in log
  - Very powerful writers can queue up writes and make sure that there are no branches or holes
  - Less powerful writers can queue up hashes over data and make sure there are only holes in data
  - Primitive writers can query servers to get "most recent" write and potentially start new branch
- Unexpected branches *only* occur when writers crash or lose state
  - Timestamps can provide "freshness" verification
  - Summary entries can provide "number of entries" or "aggregate" verification of information

Why allow branches or holes in data?

- Advantages
  - Permits tradeoffs in latency/durability/consistency
  - Allows data format to be handle a large variety of faults (failure/denial of service/etc)
  - Allows native support for versioning (think "GIT")
  - Allow temporary branches for BlockChain-like temporary versions
- Can provide "always write" semantics
  - If *any* log server is up, will be able to write data
  - Hard to know for sure about all latest entries
- Can provide stronger semantics
  - Write/Read quorums set to make latest write always visible (assuming no malicious log servers)
  - Writer and/or Reader may be forced to wait

Reasoning about the Infrastructure: Trust Domains

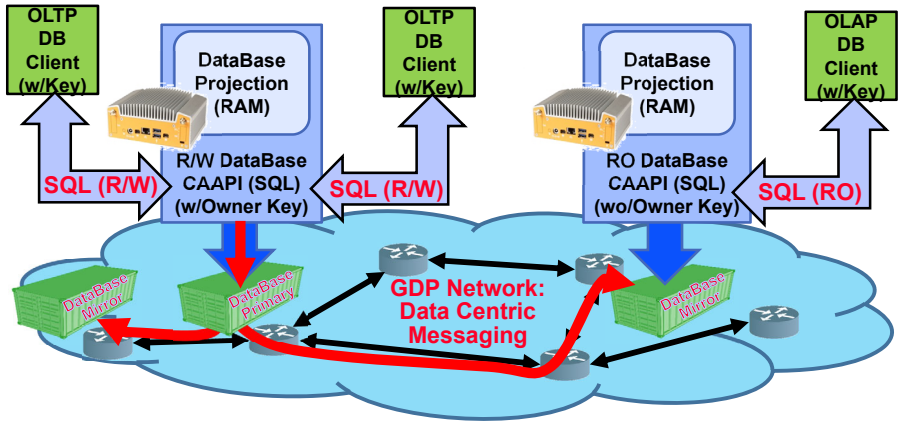


- Trust Domains:
  Groups of Resources owned by single entity
  - Reflect the ownership, trustworthiness, and degree of maintence
  - Carry unique economic, political, or incentive structure of the owner
  - Pay-for-service, Federated utility model
- Trust for:
  - Message Transport, Location Resolution, DataCapsule Service, Secure Enclave Service (SES)
  - Conversations routed according to DataCapsule owner's Trust Preferences

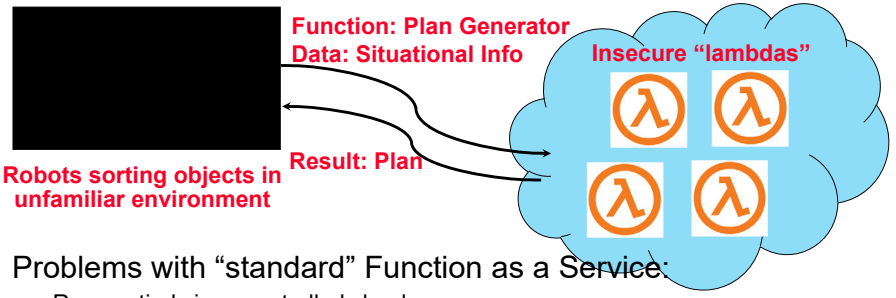
Common Access APIs (CAAPIs)

- Common Access APIs (CAAPIs) provide convenient/familiar *Storage Access Patterns*:
  - Random File access, Indexing, SQL queries, Latest value for given Key, etc
  - Optional Checkpoints for quick restart/cloning
  - Refactoring: CAAPIs are services or libraries running in trusted or secured computing environments on top of DataCapsule infrastructure
- Many Consistency Models possible
  - DataCapsules are "Conflict-free Replicated Data Types" (CRDTs): Synchronization via Union
  - Single-Writer CAAPIs prevent branches if sufficient stable storage (strong consistency models)
  - DataCapsules with branches: like GIT or Amazon Dynamo (write always, reader handles branches)
  - CAAPIs can support anything from weak consistency to serializability
- Examples:
  - Streaming storage
  - Key/Value store with time-travel
  - Filesystem (changeable sequences of bytes organized in hierarchy)
  - Multi-writer storage using Paxos or RAFT
  - Byzantine agreement with threshold admission to DataCapsules

Example #1: Using DataCapsules to build more sophisticated data access patterns (e.g. DataBase)



Example #2: Function as a Service

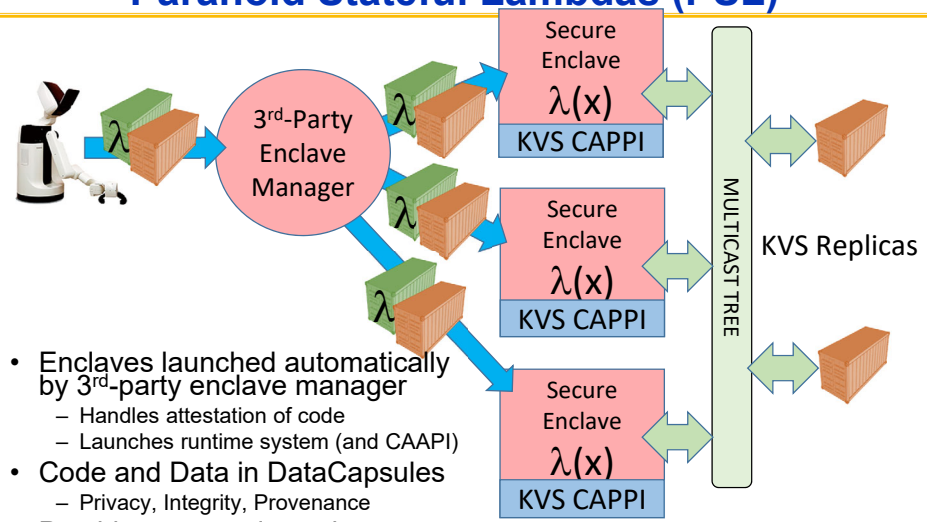


- Problems with "standard" Function as a Service:
  - Runs entirely in uncontrolled cloud
  - No explicit state: must add something else
  - Not private, leaks algorithms *and* results
- Opportunity cost: Can't exploit Edge!
  - Latency, predictability, privacy, ….

Function-as-a-Service(Faas) Cloud Service Provider ("lambdas")

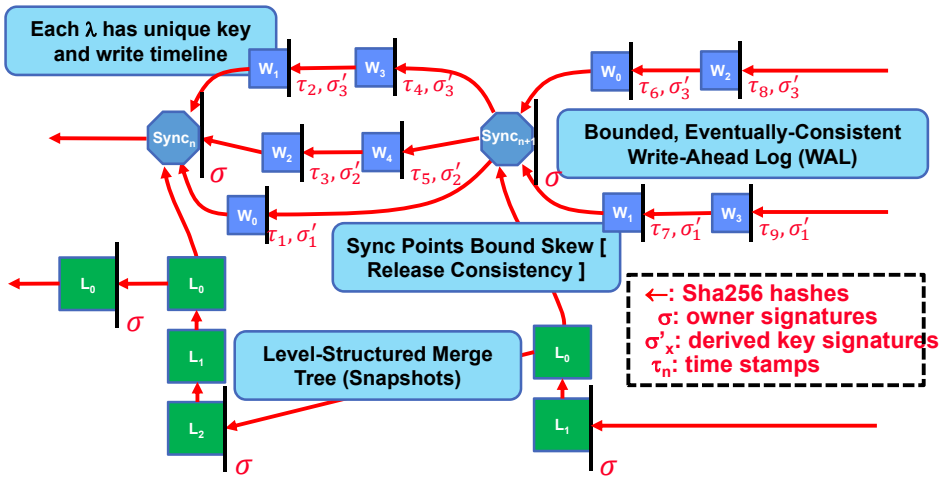
AWS Lambda Google Cloud

A better Alternative: Paranoid Stateful Lambdas (PSL)

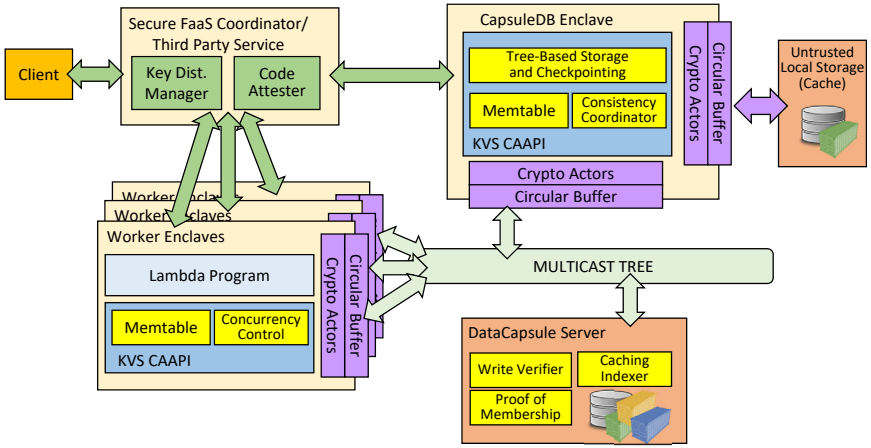


- Enclaves launched automatically by 3rd-party enclave manager
  - Handles attestation of code
  - Launches runtime system (and CAAPI)
- Code and Data in DataCapsules
  - Privacy, Integrity, Provenance
- Provides eventual consistency and release consistency

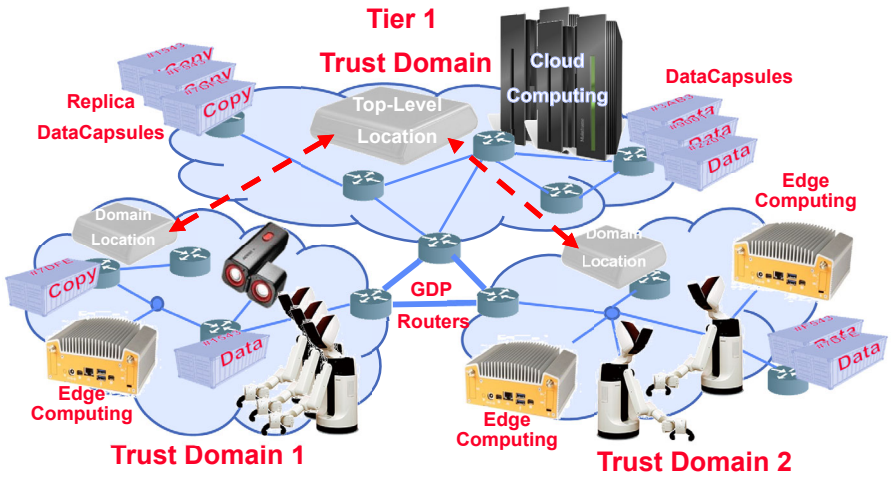
Multi-writer Model for Parallel Key-Value Store: (Inside DataCapsule)



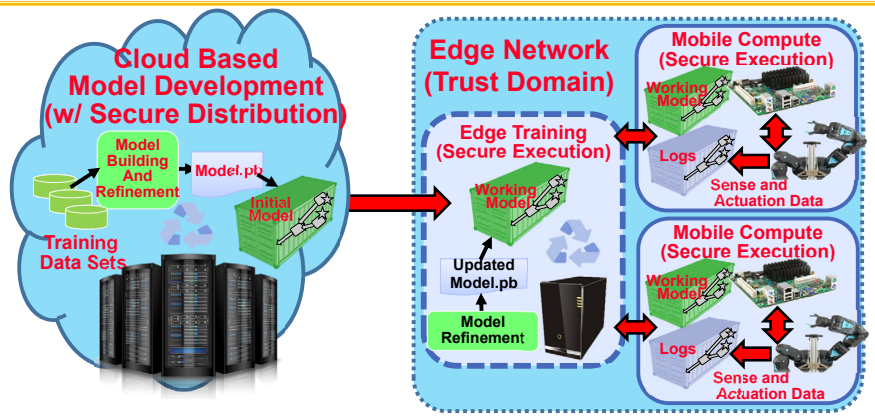
Paranoid Stateful Lambdas: Key-Value Store CAAPI for Secure FaaS



Fog Robotics on the Global Data Plane:

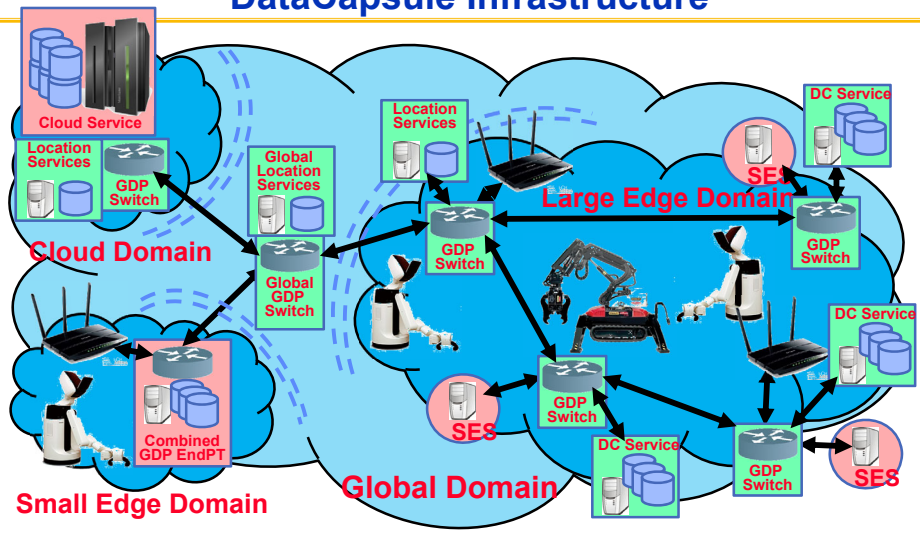


Example #3: Data Capsules for Model Delivery



- Robotic grasping model distributed in DCs
  - Intellectual property of producer (only unpacked in environments guaranteed not to leak model)
  - Refinement on the edge is updated only by authorized enclaves with attested algorithms

DataCapsule Infrastructure



Research Agenda: What is Hard?

Biggest Challenge: Convince People to Refactor their applications around DataCapsules
- Incremental Deployment encouraged via (1) overlay networking followed by (2) "native" GDP datagram routing – possibly even without IP service
- CAAPIs provide standardized storage "patterns" for naïve and domain application writers

DataCapsules provide extremely flexible storage (intended as a primitive element upon which to build a wide array of storage systems)
- The trick is to provide understandable semantics with good performance
- Consider wide range of Google storage systems (GFS, BigTable, Megastore, Spanner…)!

DataCapsule placement: Edge vs Cloud
- Placement based on Performance, Privacy Constraints, Durability Requirements, BW, QoS, ….

Replication and Failover semantics
- Basic Replication simple since DataCapsules are CRDTs (Conflict-Free Replicated Datatypes). Thus, synchronization is via union of DataCapsules is easy
- Providing quick adaptation in (routing) network as DataCapsule servers fail and recover while still providing understandable semantics is tricky

Replication in the presence of network partitions and malicious agents
- Can provide multi-writer storage using Paxos or RAFT
- Can use Byzantine agreement with threshold admission to DataCapsules

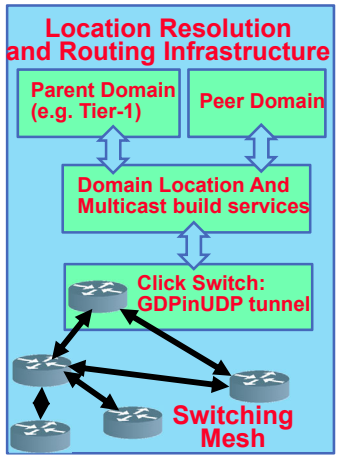
Research Agenda (con't): What is Hard?

Flat Address Space Routing is Dead, long live Flat Address Space Routing
- No physical hierarchy in the names of DataCapsules
- Each advertising certificate (Delegated Flat Name) is unforgeable (RO) and easily exported using a scalable DHT
- Using Redis key-value store for initial prototype

Adaptable, Authenticated, Automatic Multicast construction
- Multicast is an old topic, but secure, performant, multicast that respects trust domains is essential to DataCapsule/GDP
- Can leverage ideas from prior Bayeux multicast DHT work

Only Active Conversations Stored in Switches!
- Provides hope of scalability, but challenge of routing

QoS-Aware Routing problem: Efficiently routing while respecting QoS and exploiting hardware (e.g., TSN)
- Can leverage ideas from prior Brocade landmark overlay DHT work



What is short- and long-term success?

Short-term success: DataCapsules in IoT
- TerraSwarm IoT applications: DataCapsules+GDP were the Great Integrator
- For low-power devices: simple gateway talks to devices with BlueTooth, ZigBee, WiFi and speaks GDP protocols to infrastructure
- Potentially: Every camera, video, sensor, produces DataCapsules, thus enforcing provenance and consistency of output data!

Short-term success: working distributed applications and that could be experimentally tested under a variety of conditions and threats
- Fog Robotics: Robots using and generating models at the edge
- Smart manufacturing in which a 3D additive fabrication facility (at the edge) is communicating secur with a digital twin demonstrating secure and timely exchange of data between the facility and the cloud.

Long-term success: widespread usage
- Adoption of DataCapsules and the GDP as the underpinning for many devices and applications botl on the edge and in the cloud.
- Example: Adoption of DataCapsules as new standard for IoT devices and applications
- Similar to the widespread use of shipping containers in ports today

Why the Global Data Plane Again???

- Yes, you *could:*
  - *Provide your own infrastructure for everything*
  - *Provide your own storage servers*
  - *Provide your own networking, location resolvers, intermediate rendezvous points*
- But: *Why?*
  - Standardization is what made the IP infrastructure so powerful
  - Utilize 3rd-party infrastructure owned (and constantly improved) by others
  - Sharing is much harder with stovepiped solutions!
- The Global Data Plane provides *standardized infrastructure support*
  - It provides a standardized substrate for secure flat routing and publish-subscribe multicast
  - It provides a provides the ability to reason about infrastructure providers (Trust Domains)
  - It frees DataCapsules from being tied to a particular physical location
  - ⇒ Analogous to ships, planes, trains, and cranes that support shipping containers
- The GDP routes conversations between endpoints such as DataCapsules, sensors, actuators, services, clients, etc.
- Information protected in DataCapsules, but freed from physical limitations by the GDP
  - Correctness and Provenance *enforced* by DataCapsules
  - Performance, QoS, and Delegation of Trust handled by the GDP

Conclusion

- The most game-changing element of this agenda is the presence of ubiquitous, secure and mobile bundles of data: DataCapsules
  - Provably authentic and self-consistent
  - Only authorized writers can add information; anyone with possession can verify integrity
- The power of DataCapsules are in *standardization*
  - If everyone uses DataCapsules, then everyone reaps the benefits—
    No malicious information, no fake news, no breached passwords
  - Eliminate rampant "roll-your-own" philosophy that yields data breaches
- Naturally Coupled with Secure Edge Computing (Enclaves)
- Burden of standardization reduced through careful design:
  - Incremental, flat-address-space routing (no IP addresses!)
  - Efficient refactoring of communication around storage
  - Familiar storage patterns (facades): File Systems, DataBases, Key-Value Stores, Streams,…
- Exciting new applications: Robotics and Machine Learning

Is this a good paper?

What were the authors' goals?

What about the evaluation/metrics?

Did they convince you that this was a good system/approach?

Were there any red-flags?

What mistakes did they make?

Does the system/approach meet the "Test of Time" challenge?

How would you review this paper today?