UC Santa Cruz

UC Santa Cruz Previously Published Works

Title

Functional Algebraic aTomic Evaluators in Packet Routing

Permalink

<https://escholarship.org/uc/item/40f9d3h0>

Authors

Mathewson, James
Garcia-Luna-Aceves, J.J.

Publication Date

2019

Peer reviewed

Functional Algebraic aTomic Evaluators in Packet Routing

James Mathewson^{§†} and J.J. Garcia-Luna-Aceves^{§*}

[§]Computer Science and Engineering Department, University of California, Santa Cruz, CA 95064

[†]Akamai Technologies Inc., Santa Clara, CA 95054

^{*}PARC, Palo Alto, CA 94304

Email: jmathews@akamai.com, jj@soe.ucsc.edu

Abstract—Deploying new routing protocols is an expensive investment with complications from implementation and debugging on different architectures, platforms, network constraints, and algorithmic interpretations. We introduce an alternative to strict algorithmic programming of protocols by introducing Functional Algebraic aTomic Evaluators (FATE). FATE is an information-processing engine that evaluates information, using algebra, and acts upon said results. FATE is designed to be highly configurable and flexible to allow rapid development on any platform, or necessary changes. FATE uses an XML configuration file to determine which path is optimal. Changes to routing constraints can be met by changing the configuration file, as oppose to redesigning, testing, and debugging a new algorithm. The FATE platform is designed not only for rapid development, but for accuracy in duplicate functionality on different platforms, and ease of use.

I. INTRODUCTION

Developing a new network routing protocol can be an laborious process complicated by the constraints on how information is shared. Each of these constraints affect the chosen routing algorithm: how is collision detected (if detected at all), and how is it handled? What is the best route, hop distance vs link bandwidth? Priority content vs low priority content, how is it handled? The answer, typically, is "It depends". Depending on what is desired, a longer optical fiber route may be preferred to over-the-air transmission. Battery life for mobile devices/sensors may limit what needs to be routed. FATE is similar to an FPGA; it is not as fast as a custom ASIC, but it has many advantages: it allows reprogramming in the event the internal algorithm is not optimal; it allows immediate testing of algorithms, as opposed to waiting for custom silicon to be delivered. When a potential design is hindered by multiple parameters, creating multiple versions of the software, each version having different dependencies, will create unnecessary hardship in comparison of results, validation of algorithm, and later improvements. Creating new algorithms, and comparing them to existing research has many problems, including reproducibility [5] [13].

FATE evaluates information, from contextual metadata, to decide a course of action, based upon the evaluated results. In essence, instead of a dependency upon a fixed algorithmic implementation, FATE makes decisions based up algebraic atomic evaluators. Each evaluator performs an atomic evaluation of information, which reflects the Linux philosophy of "Doing one thing, and do it well". The entire evaluation

formula consists of the atomic evaluators, connected by algebraic expressions, to give a higher level evaluation (e.g. $\text{MIN}(\text{FN1}, \text{FN2}, \text{MAX}(\text{FN3} * 0.6 + 0.4 * \text{FN4}, 0.4))$). FATE, in networking, can be used to evaluate which content to store/evict in a cache, determine which egress port offers the best performance (in terms of networking constraints), or which content to prefetch, etc. The purpose of FATE is two fold. The first is to allow rapid changes to a given algorithm, due to algorithmic changes, erroneous assumptions, or different constraints. The second purpose is duplicatability of results regardless of architecture. FATE is agnostic to the types of networking environments (hardware or various simulators), or kernel/operating system implementations. FATE, as presented, allows faster development for custom routing protocols, and ensures reproducibility when compared on different platforms.

II. RELATED WORK

A. NS3 Network Simulator

NS-3[3] is one of the most popular network simulators, with over a thousand papers published on its platform. NS3 is event driven, written in C++11 (same as FATE), to schedule networking events. NS3 also models real world interactions, such as error rates on various mediums, and collisions.

B. ENCODERS

ENCODERS[6], [12] (Edge Networking with Content-Oriented Declarative Enhanced Routing and Storage) is an SRI implementation of the PSIRP ICN models, based upon Haggie[10]. PSIRP and Encoders both use a bloom filter based Pub-Sub model, disseminating interests to neighbor nodes, and those neighbors return matches to the data. ENCODERS used a very early iteration of FATE called Utility Networking (Utility Caching-Prefetch, and Utility Forwarding), which evaluated content, and based upon that value, made a decision. In ENCODERS, Forwarding was fixed to several schemes, and was done via an early immature version of Utility Forwarding, Prophet[7], Direct[11], or Epidemic[14].

C. Traditional Routing

Traditional routing, such as IP (Internet Protocol), use a routing table, consisting of subnets, which dictate the interface, gateway and next-hop neighbor for each forwarded packet. The tables are populated by various routing protocol to determine the best egress port, for packet delivery; typically shortest hop

distance or largest path bandwidth. The tables are populated by various routing protocols, including but not limited to: OSPF (Open Shortest Path First) [9], RIP (Routing Information Protocol) [4], EIGRP (Enhanced Interior Gateway Routing Protocol) [1], OLSR (Open Link State Routing) [2], and static routing. When a packet is ready to be transmitted, the Layer 2 medium is checked for in-progress packet transmissions. Typically, wired networks use CSMA/CD (packet collision) and wireless networks use CSMA/CA (use RTS/CTS or Ready/Clear signals to control when packets are sent) to avoid packet congestion/collisions in the network.

III. FATE FORWARDING IMPLEMENTATION

FATE resolves system dependencies, such as timers or hardware (such as GPS), by using its own functions, wrapped and translated (if necessary) from the correct architectural implementation. Thus, a timer on the NS3 simulator will use the event-driven timer available in NS3, but in a Linux implementation, use the Linux timer POSIX methods. Mentioned for completeness, FATE uses a flexible packet framework, which uses a type-name-value tuple in the packet. FATE does use uniquely named information, or named data, to identify each unique chunk of data, allowing access to all the packet attributes by name.

FATE is organized in a top-down format, where the 'Node' module (III-B) handles specific purpose modules. Specific purpose modules can be any purpose, with the intent to have the specific module do its own job, for easier maintenance, verification, and testing. Some modules are forwarding, caching, discovery, security, et al.

Each utility is an atomic, algebraic function, which evaluates the content, or returned value, and returns a normalized scalar ($[0, 1]$). This allows a very flexible and powerful method to evaluate information. The module will take a valuation of the content, and perform an action on it (e.g. Caching will store or evict content, forwarding will decide which packet for which egress port and next hop neighbor, security will evaluate the packet for trust-worthiness, etc). In this paper, we concentrate on the forwarding aspects of FATE.

FATE uses a modified BSD license. The license, makes the code free to use, with the exception of giving credit when FATE is used in any manner.

A. Functional Algebraic aTomic Evaluators

The concept of FATE is to evaluate information (packet attributes, network conditions, or physical (PHY) properties), and perform an action, based upon said result. In order to evaluate a result, atomic algebraic functions are used. Each function can be an aggregate (such as minimum, addition, etc), or an atomic evaluator.

1) *Aggregation Functions*: FATE supports several aggregation methods, all take one or more inputs, and returns an appropriate result. Below is a partial listing of available aggregation functions:

MIN : MINIMUM(a,b,...,z) return the minimum value of its inputs.

MAX : MAXIMUM(a,b,...,z) return the maximum value of its inputs.

ADD : ADDITION(a,b,...,z) return the sum of all its input. The sum may be greater than 1.0, and may require scaling.

MULT : MULTIPLICATION(a,b,...,z) return the product of its inputs.

2) *Atomic Functional Methods*: FATE supports several atomic functional methods, below is a partial listing. Each atomic function may be stateful, but the state is exclusive to each instance of the function. Atomic functions evaluate a specific attribute, functionality, algorithm response, or statistical method, with a specific purpose, to provide an evaluation based upon its functionality (as an example, certain algorithms are based upon several or multiple parameters; whereas FATE is based upon the principle to have many singular functions do the evaluation, then weighted based upon the appropriate aggregate function). Many functions are generic, with configuration options to allow them to be aliased to specific functionality. The following is a subset of atomic algebraic functions, currently available:

CONSTANT: A constant value, typically used with multiplication, e.g. $0.4 * \text{TOS}$ (Type Of Service).

PKT_ATTRIBUTE: This function relays a value of a specified packet attribute. Examples include TTL (which returns a '1' if TTL is 1+, but '0' if it is zero), TOS/COS (where a matching attribute value returns a predefined value, e.g. TOS of '3' returns '1.0', etc). Some of the functions are generic, applied to many fields (e.g. TOS matches a specific value, it returns '1'. It depends on multiplying by a constant to give a weighted value, such as $0.4 * \text{TOS}$. While the generic function uses a named field (all field attributes are accessed by name), they are, typically, written in a straight forward manner to identify their purpose. Thus, 'TOS' is shorthand for PKT_ATTRIBUTE("TOS", 3).

TABLE_ATTRIBUTE : ATTRIBUTE(node/PHY-name, name) is used when a function needs to access a table to determine a value. Examples of this are properties of the PHY (speed or properties, such as secure fiber optical cable vs wireless broadcasting), or a nodes' power measurement (e.g. how much battery is left). Other traditional attributes may be accessed from an internal node table, including if the medium is busy, PHY is transmitting, or measurements of network congestion (if available). Typically, discovery (or HELLO) packets are used to help fill in tables, such as new neighbors, or monitor how often collisions occur.

TUPLE_NORMALIZED : TUPLE_NORMALIZE(PHY, Source, Destination, value, FN()) Table of tuples, which return a value. Typically, hop-counts for the specified PHY, and how many hops from 'Source' to 'Destination', are used as a key. Any tuple property can be modeled, but, for this paper, it is aliased to HOPCOUNT(). The actual normalization function is dictated by passing in a function, FN(). This allows a choice of linear, logarithmic, or other normalization of counts.

Packet Attributes: FATE packets can carry many attributes, most of which can be evaluated. There is an exception to this, currently being used to forward IP packets. By using a chain

(list) of intermediate destinations, and allow the existing layer 3 (typically IP) to route the packet. FATE has its own packet format, and can be encapsulated within any L2/L3/L4 packet (or it can be left a pure FATE packet). But, this attribute can be used to encapsulate a list of Addresses (A,B,C,D). When a packet (such as IP) is sent and arrives at the correct destination, this field is checked, and the destination packet is change. E.g. if a packet has a destination of E, when it arrives at E, it is replaced with 'A' (which is popped off the attribute queue). When the packet is received by 'A', it is forwarded to 'B' (as the new destination). There are multiple uses for this feature, but it does not evaluate as other atomic functions. This feature is used to query specific nodes, off-path caching, share information between specific network nodes, and avoiding known down network pitfalls. The routing used is the original routing protocol, but simple forced to route multiple times, to collect or share information.

B. Modules

Fig. 1. Simple Forwarding representation of $\text{MIN}((0.8*\text{HOPCNT}+0.2*QoS), \text{PHYFREE}, \text{TTLVALID})$

FATE is structured as a tree illustrated in Figure 1. Atomic functions are the leaves of the tree and algebraic aggregate functions are tree branches. The information packet is passed down and evaluated at each leaf. The results then move along the aggregated branches towards the root to give an evaluated result.

To easily model this, we need three main (atomic) functions to forward, in this simple example: We take an evaluation of the HOPCNT (hop count to the destination, from a specific neighbor node and PHY), weighted at 80%, and the QoS (Quality of Service, typically TOS field in IP) at 20% weight (summing them). To prevent sending a packet when the TTL has expired, we do a MIN (minimum) with TTLVALID ('1' if the TTL field is 1+, otherwise '0'); note the figure shows it being pre-decremented (some FATE utilities can modify the packet fields/attributes). In addition, we do not wish to transmit the packet when the PHY is busy, or the timer is still counting

down from a packet collision, such as CSMA/CD or /CA (represented by PHYFREE). Based upon the evaluation, it is up to the final evaluation to decide if the packet is forwarded, dropped, or must wait in queue.

Fig. 2. Utility Forwarding Tree representation of

When FATE evaluates a packet for forwarding, it evaluates against every PHY-Neighbor pair (against the destination node, not shown). As shown in figure 2, UF1 has 1 neighbor node via bluetooth, UF2 has two WiFi neighbors, and UF3 and UF4 each have a single wired neighbor. Notice the UF1 and UF2 share the same neighbor, but using different PHYs. For each packet, the evaluation occurs on the UF1-Node2, UF2-Node2, UF2-Node3, UF2-Node4, UF2-Node5 pairs. Thus, in this case, for a single forwarding, 5 evaluations are made, and the highest rated evaluation wins. FATE allows a minimum threshold to send. In case there is more than a single packet to send, the first packet is sent out, and that PHY (and its neighbors) are removed from evaluation. Thus, if a queue of 2 packets are waiting to be routed, and the first is transmitted on UF2, the next packet is evaluated on UF1, UF3, and UF4. If any of those meet the minimum threshold to send (the threshold is set in the XML file), it will be sent on another PHY, to allow more efficient load balancing. As an example, WiFi might be preferred, but bluetooth could be acceptable for packet transmission. FATE configures at the PHY level, sharing the same configuration for each PHY-Neighbor pair, but does not require all PHY configuration files to be the same (allowing flexibility in WiFi vs Ethernet collision avoidance/detection). When each neighbor is compared, the table lookup utility is used to identify (as a normalized vector) the distance to the intended destination.

IV. SAMPLE RESULTS

FATE is currently available, but the full integration with NS3 simulator is not yet complete (caching is complete, with partial routing). FATE is missing the NS3 integration to retrieve hop-counts (from IPv4), PHY speed, and collision rate from the NS3 simulator.

Figure 3 illustrates a sample network using Table I for the PHY properties of Node B and Table II for the neighbor $\leftrightarrow \{destinationhopcountperphy - pair\}$.

As stated in Table I, the value of the PHY network speed is logarithmic to the largest value (1Gbps). In a similar manner, the hop count is reduced to a normalized value between [0,1]. The value of a destination 1-hop away is '1.0', while three hops away is '.034'.

Fig. 3. Representation of FATE with PHY-Neighbor pairs

Using an algebraic formula of $HOP * SPEED$ for route B to F, returns three values, the Bluetooth route (1 hop, lower speed; $0.23 * 1 = 0.23$), or the wired route (3 hops, 1gbps speed; $0.34 * 1 = 0.34$). From this formula, it is preferable to send packets over the longer but faster wired route. Another example is from node B to C, which gives a value, of 1 hop via WiFi of $1.0 * 0.67$, as opposed to a wired route (2 hops) of $0.67 * 1$. In this instance, both have an equal rating (ties can be broken by random determination, or by additional algebraic criteria).

TABLE I
SAMPLE PHY TABLE SETUP FOR NODE B

PHY	Range	Speed	SpeedValue (logarithmic)
1-Wire	inf	1gbs	1
2-Blue-tooth	5m	5mbs	0.23
3-Wireless	60m	100mbs	0.67

TABLE II
SAMPLE PARTIAL HOP TABLE SETUP FOR NODE B

PHY	Neighbor	Dest Node	Hops	HopValue
1	A	A	1	1
1	A	C	2	0.67
1	A	D	3	0.34
2	F	F	1	1
2	F	E	2	0.67
3	C	C	1	1
3	D	D	1	1
3	D	E	2	0.67

Or, using TOS values to determine priority in a congested system, where TOS of 0 returns '0.2', and TOS (Type of Service) of 7 returns '1.0', we can a simple $TOS * HOPS$. This allows short path packets to be delivered despite their TOS value, while longer packet paths are served only by higher value TOS. Assuming a maximum of 20 hops, a TOS(0), going 1 hop, will have the same value as a TOS(7) packet going 4 hops ($TOS * HOPS$; $0.2 * 1.0$ for short path, $1.0 * 0.2(4/20)$).

There is no restriction to HOW the packet gets evaluated. To always give priority to TOS packet, routing can occur by TOS, then HOPS, e.g. $TOS * 0.8 + HOPS * 0.2$, or $\min(TOS, HOPS) * SPEED$. Obviously, these values are chosen to give a sense of how a route would be selected, by evaluation, not an actual forwarding algorithm per se.

V. CONCLUSION

Reducing algorithmic functionality to specialized atomic functions greatly increases the flexibility and power of new implementations of routing protocols. As each atomic evaluation function is created, it adds to the basic building blocks for succeeding algorithms, and enables designers to focus on rapid and efficient algorithmic development. The plug-and-play aspects of FATE significantly reduce development time, and turn complex algorithms to simple algebraic expressions (configured by XML). FATE forwarding is meant to be a consistent cross platform and allow rapid development. It still keeps internal state and processes each packet to every PHY-Neighbor (to destination) pair. Typically, FATE uses more CPU and is slower than the equivalent optimized algorithm. FATE is implemented under a modified BSD license, with the only requirement of giving credit for usage of the code or concepts therein. The code is available online [8].

ACKNOWLEDGMENTS

This work was supported in part by the Jack Baskin Chair of Computer Engineering at UC Santa Cruz.

REFERENCES

- [1] R. Albrightson, J.J. Garcia-Luna-Aceves, and Joanne Boyle. EIGRP—a fast routing protocol based on distance vectors. 1994.
- [2] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). Technical report, 2003.
- [3] NS-3 Consortium. Network simulator ns3. Online; accessed 13-October-2015.
- [4] C.L Hedrick. Routing information protocol. Technical report, 1988.
- [5] S. Kurkowski et al. Manet simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, 2005.
- [6] D. Kutscher et al. Information-Centric Networking 3 (Dagstuhl Seminar 14291). *Dagstuhl Reports*, 4(7):52–61, 2014.
- [7] A. Lindgren et al. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, 2003.
- [8] J. Mathewson. Fate - functional algebraic atomic evaluators. <http://github.com/jlmathew/Fate>. Online.
- [9] J. Moy. Ospf version 2. Technical report, 1997.
- [10] J. Scott et al. Hagggle: A networking architecture designed around mobile users. In *WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services*, pages 78–86, 2006.
- [11] I. Solis and J. J. Garcia-Luna-Aceves. Robust content dissemination in disrupted environments. In *Proceedings of the Third ACM Workshop on Challenged Networks*, CHANTS '08, pages 3–10, New York, NY, USA, 2008. ACM.
- [12] SRI. Edge networking with content-oriented declarative enhanced routing. Online; accessed 13-October-2015.
- [13] I. Stojmenovic. Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions. *IEEE Communications Magazine*, 46(12):102–107, 2008.
- [14] A. Vahdat et al. Epidemic routing for partially connected ad hoc networks. Technical report, Technical Report CS-200006, Duke University, 2000.