An Internet-Spanning Content Distribution Mechanism for IoT

Griffin Potrock

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-56
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-56.html

May 11, 2018

Acknowledgement

I'd like to thank the SwarmLab for supporting me through my degree. In particular, I'd like to thank Professor John Kubiatowicz for serving as my adviser and for helping shepherd this research. I'd also like to thank Nitesh Mor for many design meetings and significant contributions to the GDP sections of this thesis. I am grateful to Eric Allman and Richard Pratt, who served as frequent sounding boards. Thanks is due to Ken Lutz for his advice in the process of producing this work. Dylan Dreyer also deserves credit for contributions implementing and analyzing the simulations. I would also like to thank Professor Ion Stoica for his feedback on the project's direction. Finally, I would like to thank the faculty reviewers of this work, Professors John Kubiatowicz and John Wawrzynek.

An Internet-Spanning Content Distribution Mechanism for IoT

by

Griffin Potrock

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Kubiatowicz, Chair
Professor John Wawrzynek

Spring 2018

The thesis of Griffin Potrock, titled An Internet-Spanning Content Distribution Mechanism for IoT, is approved:

Chair _____ Date _____

_____ Date _____

University of California, Berkeley

An Internet-Spanning Content Distribution Mechanism for IoT

Abstract

An Internet-Spanning Content Distribution Mechanism for IoT

by

Griffin Potrock

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor John Kubiatowicz, Chair

Low-cost, Internet-connected devices are rapidly proliferating in a computing mega-trend known as the Internet of Things (IoT). While the IoT offers great opportunities, from smart cities to smart homes, it also offers many new computing challenges. These challenges include handling larger numbers of devices; handling more upstream and inter-device communication; and managing the secure storage and distribution of rapidly increasing amounts of data.

The Global Data Plane (GDP) project seeks to re-architect the networking infrastructure of the Internet to accommodate these trends. The GDP relies on replicated, append-only logs. In addition to being a durable data store, these logs are often used as a single-writer publish/subscribe mechanism.

This thesis proposes new mechanisms for adapting publish/subscribe to the networking challenges of IoT. We detail design choices for a new overlay-based multicast system, the Secure Content Distribution Tree (SCDT), that are both novel and thoroughly grounded in existing literature and experience to ensure viability. We also propose new, scalable mechanisms for providing reliability in such a system. While our evaluation focuses on demonstrating the viability of our multicast architecture and reliability mechanisms through simulations, we also include discussions on security and deployment.

Contents

List of Figures

List of Tables

Acknowledgments

Many people contributed to the completion of this thesis. I'd like to thank the UC Berkeley SwarmLab in general and the SwarmLab Global Data Plan group in particular for supporting me through the research process. In particular, I'd like to thank Professor John Kubiatowicz for his architectural guidance, for serving as my adviser, and for helping shepherd this research to completion. I'd also like to thank Nitesh Mor for many long design meetings and his sage advice on graduate studies, as well as significant contributions to the GDP sections of this thesis. I am eternally grateful to Eric Allman and Richard Pratt, who were both critical in helping me clarify the structure of SCDTs and served as frequent sounding boards. Thanks is also due to Ken Lutz for his advice and guidance in the process of producing this work. Dylan Dreyer also deserves credit and praise for substantial contributions in implementing and analyzing the simulations presented in this work. I would also like to thank Professor Ion Stoica for his feedback on the project's direction. Finally, I would like to thank the faculty reviewers of this work, Professors John Kubiatowicz and John Wawrzynek, for taking the time to review this thesis.

Chapter 1

Introduction

1.1 Background

The Internet of Things is a computing macrotrend poised to change the way we interact with computing environments and reshape the Internet. While the push toward cloud computing has lead to increasing centralization of the Internet into a handful of data centers, the proliferation of IoT devices is pushing computation and data flows back toward the network edge.

IoT applications may be worth up to $11 trillion by 2025. However, 40% of this value relies on coordination between IoT systems [36]. Developers face any a number of challenges in capturing this value. An effective IoT deployment cannot simply be a direct connection between every individual IoT device and a cloud data center [60]. Round trips to the cloud are inefficient in latency, bandwidth, and network , limiting scalability and imposing deployment constraints. IoT devices are often embedded, low-power devices with low duty-cycles, making ensuring reliability and durability of data at best an unnecessary energy drain and at worst a debilitating constraint. Furthermore, routing to and utilizing the cloud comes with a number of privacy and security risks.

Latency, Bandwidth, and Scalability

In many cases, IoT devices can only realize their potential when they are able to distribute their data to thousands or millions of subscribers efficiently. For example, a temperature sensor might need to publish current temperature readings to every HVAC system in a neighborhood, or an air quality sensor might need to send pollution warnings to citizens in a wide area.

Many IoT applications involve real-time latency constraints, which almost entirely preclude going to the cloud to access data. A traditional option is for the device originating the record to store and retransmit it to interested nearby devices that fail to receive the original transmission. This is a poor option given the constraints of such end-devices. If neither

the cloud nor the original device are a reasonable source of caching and retransmission, that means that responsibility must be pushed elsewhere in the network.

Since many IoT applications involve nearby devices intercommunicating, round trips to the cloud make little sense. Such a solution would place undo stress on the border gateways of the network, requiring beefy links to the Internet. This is an unnecessary expense and a serious problem for remote deployments.

Device Constraints

The uniformity of the phrase "Internet of Things" obscures the massive variety of devices and software that will be deployed in the IoT, not only across manufacturers and developers but also between different versions deployed at different times. Achieving consistent performance is difficult in the presence of such heterogeneity. Some popular controllers such as Raspberry Pi [48] and BeagleBone [9] are powerful enough to run full desktop Linux distributions. Others, such as Telos B [46], which remains popular in the wireless sensor network research community, focus on minimizing power consumption. Even low-power motes have a variety of choices when it comes to operating systems, including TinyOS [33], Contiki [18], and RIOT [5]. Further, such motes may only be able to transmit occasionally due to power constraints, such as motes that use power harvesting techniques.

Developers currently looking to build heterogeneous IoT systems must develop solutions that can coordinate across an ever increasing mix of hardware and software deployed in the field. The low duty cycle of some IoT devices can make direct inter-device communication difficult.

In many cases, IoT devices will also have to ensure the reliability and durability of their data. Having all data subscribers communicate directly with the publisher, such as with TCP, is essentially impossible due to inconsistent timing and a lack of sufficient processing power and bandwidth on virtually all IoT devices to service heavy traffic.

Privacy and Security

Security requirements can impose significant constraints on any networking protocol for the IoT. Many applications will require data confidentiality, necessitating that all data be encrypted. This encryption scheme must be scalable to communication with thousands of devices.

Encryption alone, however, does not preclude side-channel attacks or traffic analysis attacks [45]. One example might be a device that writes an encrypted "open/close" command to a door, allowing anyone who can snoop on the encrypted traffic to determine when someone enters/exits the building without decrypting the data. Corporations in particular frequently do not want to entrust their proprietary data to external storage or allow it to be routed outside their corporate network, even when it is encrypted. Data regulations in some countries restrict what data can flow across international borders [14]. Even within

	IP Multicast	Overlay Multicast
Incrementally Deployable	No	Yes
Easily Support Firewalls/NATs	No	Yes
Network Stress	Lower	Higher
Average Stretch	Lower	Higher

Table 1.1: Comparison of IP multicast and overlay multicast.

countries, any IoT data security scheme must allow some restriction on where data is allowed to flow, or set up secure, noise-injected channels between trusted nodes.

1.2 Related Work

Our solutions build upon the large body of academic literature and industry experience in multicast. Multicast is fundamentally a simple concept: rather than sending packets to individual destinations, the network uses intermediate routers as fanout points to reduce the strain on any one router. Unfortunately, this concept has seen limited adoption due to a number of implementation and deployment issues. There are two fundamentally different categories of multicast schemes: IP multicast and overlay multicast. Either can be used in a publish/subscribe system.

IP Multicast

IP multicast is a network-level multicast concept that has been a popular research topic since at least the 1990s. Despite the uniformity implied by the name, there is no one single IP multicast protocol or technology. Rather, IP multicast instead refers to a collection of protocols. In general terms, these protocols rely on constructing forwarding tables at individual routers that map an IP multicast address to a series of next-hop routers. IP multicast addresses are specified in RFC 1112 [15]; specifically addresses ranging from 224.0.0.0 to 239.255.255.255 are pointed to zero or more end-hosts.

Perhaps the most common IP multicast deployment involves Protocol Independent Multicast (usually Sparse Mode) [19] and Internet Group Management Protocol (IGMP) [21]. Although they operate at the network level, these protocols operate above the protocols that actually construct IP forwarding tables. Therefore, they can be used in conjunction with most routing protocols, such as OSPF [42], IS-IS [29], and RIP [35] - hence the "Protocol Independent" portion of the name.

In brief, PIM-SM works by having routers with downstream clients send Join/Prune requests towards a designated Rendezvous Point (RP) and using these requests to build the forwarding tables. Data is then multicasted by having each router forward the data on all interfaces that have downstream clients in the multicast group.

There are any number of alternative and supplementary protocols in the IP multicast space. PIM Dense Mode (PIM-DM) [1], Border Gateway Multicast Protocol (BGMP) [55], Multicast Open Shortest Path First (MOSPF) [41], Distance Vector Multicast Routing Protocol (DVMRP) [56], Core Based Trees (CBT) [7], and Ordered Core Based Trees (OCBT) [51] all fill similar niches with varying degrees of success. PIM can also be supplemented with protocols like Multicast Source Discovery Protocol (MSDP) [37], which interconnects PIM-SM domains. Multicast Listener Discovery Protocol (MLDP) [28] is essentially the IPv6 version of IGMP.

A number of reliability schemes have been implemented on top of IP multicast, and are overwhelmingly based on negative acknowledgments (NAKs). NACK-Oriented Reliable Multicast (NORM) [2] handles reliability by asking receivers to send a negative acknowledgment to request retransmission when a missed packet is detected. Pragmatic General Multicast (PGM) [52] also uses nacks but trades off reliability guarantees for performance. Scalable Reliable Multicast (SRM) [23] includes stronger locality principles: receivers recover by multicasting a repair request, and the missing data is retransmitted by any host that has received the packet. Excessive repair requests/retransmissions are suppressed using exponential backoffs.

None of these protocols have seen much deployment outside of individual organization networks, let alone an Internet-spanning deployment that would be needed in an IoT world. The biggest problem has always been the deployment of multicast-capable routers. Since IP multicast is network-level, generally all or at least most routers in the network must be able to "speak" the required protocols. Similar to IPv6, which despite substantial effort reached just 10% deployment by its 20th anniversary [10], IP multicast cannot be fully effective until a large portion of the Internet adopts it, but few ISPs want to invest in a protocol with vague future returns. This is the primary reason IP multicast has seen some limited deployments, such as in corporate networks where the deployment can be controlled by a single entity, but has not been widely deployed in the broader Internet. Other limiting factors on IP multicast include but are not limited to difficulties handling interdomain routing (and who will pay for it); problems handling NATs and firewalls; and security/authorization challenges [17].

Overlay Multicast

Overlay multicast (also called Application Level Multicast or Application Layer Multicast) schemes arose to mitigate some of the problems faced by IP multicast schemes. Overlay multicast utilizes the same fundamental concept as IP multicast, using intermediate routers as fanout points to improve network efficiency. As the name suggests, overlay multicast schemes operate on overlay networks [34]; they are application-level, rather than network-level, protocols.

The biggest advantage of this approach is that it eliminates the deployment problems faced by IP multicast; "multicasts" actually take the form of a series of unicast transmissions to specific destination routers, routed over IP, which then further propagate the transmission. Therefore, a handful of hosts running overlay multicast software can be deployed and reap

	GDP	RabbitMQ	Kafka	ZeroMQ
Data Distribution	Push	Push	Pull	Pull
Data Structure	Append-Only Log	Queue	Append-Only Log	Queue
Data Filtering	No	Yes	Yes	No
Edge-Ready	Yes	No	No	No

Table 1.2: Comparison of publish/subscribe systems.

(some of) the benefits of multicast without having to deploy IP multicast-enabled hardware throughout the network.

One of the biggest challenges with overlay multicast is that the notion of neighboring nodes is not as intuitive as it is in IP multicast. For instance, simply routing a join request towards a rendezvous point (as PIM-SM does) does not guarantee that the packet will ever encounter a router running the overlay multicast software before reading the RP, reducing such a scheme to little better than unicast. In addition, overlay multicast is inherently less efficient than IP multicast because overlay multicast does not fully consider the underlying network the way IP multicast can.

There have been a number of influential overlay multicast implementations. Scribe [12] builds a multicast tree on top of Pastry [49], a Distributed Hash Table (DHT) implementation with locality properties. By routing along Pastry towards a rendezvous point, Scribe constructs a multicast tree from the union of the routes along the DHT. Overcast [30] takes a different approach. Joining nodes will contact the root of the overcast tree and sample the connection bandwidth. The joining node then begins a series of rounds in which it will sample the current parent node's children and attach itself to the closest child that does not significantly reduce bandwidth. Narada [13] generates a connected graph among the nodes called a mesh and constructs spanning trees from there.

Publish/Subscribe Systems

The pub/sub architecture [20] is a frequently used communications model for distributed applications. As the name suggests, subscribers register interest in specific events, and are notified of relevant events created by publishers. While the model is simple, there are many different implementations that favor different design constraints. Dedicated pub/sub systems include RabbitMQ, Apache Kafka, and ZeroMQ.

RabbitMQ [3, 4] and Kafka [31] are both pub/sub infrastructures. There are several similarities, including having a similar architecture based on queues and message brokers. The biggest fundamental difference between the two is that RabbitMQ brokers push data to subscribers; in Kafka, clients must request data from the brokers. Both are primarily centrally deployed (e.g. in data centers) and are mainly scaled by increasing the number of brokers.

ZeroMQ [27] takes a fairly different approach. When configured to use multicast, subscribers attach to the multicast tree via multicast switches; publishers send data to these multicast switches to forward through the network. These switches use Pragmatic General Multicast (PGM) [52], discussed briefly in section 1.2, to distribute data. Unfortunately, ZeroMQ inherits many of the issues associated with PGM. ZeroMQ cannot generally be deployed on top of an existing network because of its reliability on IP multicast. Publishers have no way to determine when subscribers join, fail, or reconnect. Subscribers have no ability to communicate with publishers to control the rate of messages; they must either receive published data at full speed or drop packets.

All three of these systems exist in data centers and are not aimed at deployment on the edge from either a scalability or security perspective. See chapter 2 for more on the edge-focused pub/sub paradigm offered by the GDP and for our arguments on the advantages of this paradigm over these existing pub/sub systems.

1.3 Motivation

This thesis argues for a new approach to multicast, primarily in order to enable a more efficient publish/subscribe mechanism for IoT. Decades of work on IP multicast, however, have produced lackluster results in real-world deployments, with the biggest roadblock being the inability to incrementally deploy a multicast service. Overlay multicast schemes were created to solve these problems.

However, existing overlay multicast schemes remain limited in their use cases. None target the level of scale necessary for the Internet of Things. When it was introduced, Overcast [30], for instance, was not evaluated on real deployments of significant size, and only simulated on up to 600 nodes. Multicast groups in the IoT could grow to thousands or even millions of nodes. Even without considering such large groups, with the number of Internet-connected IoT devices surpassing 31 billion [40] in 2018, smaller multicast groups will often be sharing the same infrastructure, so multicast mechanisms must still be relatively lightweight and efficient.

Further, existing overlay multicast mechanisms are targeted at a non-mobile publisher maximizing throughput to subscribers. Taking full advantage of the Internet of Things will require pushing computation and networking to the edge. In many cases, this will mean an IoT multicast scheme must rapidly adapt to a mobile publisher. Perhaps even more significantly, developers will need the ability to rapidly push data to local devices. IoT devices in the same vicinity may need to interact with each other in real time, such as a smart home that turns on a light when a user opens the door or streams security camera footage to the living room TV.

Devices further away are less likely to have real-time dependency on the data. This leads us to propose a shift: that an IoT multicast scheme should prioritize delivering data to local subscribers over subscribers further away. We further argue that, given the relatively small or infrequent messages published by many IoT devices (e.g. a temperature sensor publishing

readings once per minute), an IoT multicast should prioritize latency over bandwidth. Of course, latency is determined not just by the route packets take but also by the stress on the network, the fanout at any individual node, and the quality of links (if the scheme is reliable, lossy links will require more retries). We argue that the metric to optimize is *stretch*. Stretch is defined as:

$$\frac{Actual\ Route\ Latency}{Optimal\ Route\ Latency}$$

We measure stretch for a given node in practice using:

$$\frac{(Node\ Latency\ to\ Parent) + (Parent\ Latency\ to\ Publisher)}{Node\ Latency\ to\ Publisher}$$

By setting a worst case stretch value, a multicast scheme can force traffic to route through non-optimal routes in order to improve fanout and reduce stress, while simultaneously preventing long, snaking multicast trees that reduce latency.

Stress on a given link is defined as "the number of identical copies of a packet carried by a physical link" [13]; *average stress* is the average of stresses across all physical links in the network. This metric is already established in the literature as a relevant measure of the efficiency of application-level multicast.

For the purpose of the remainder of this paper, we generally refer to the "latency" of a given node as the one-way transmission time from the root of the multicast tree to the given node. One of the advantages of our simulation environment, described in chapter 3, is that we do not have to perform clock synchronization or use round-trip times to determine latency. We are primarily interested in this particular latency because it represents the path data will generally take from the publisher to subscribers.

Unless otherwise noted, the "root" of the multicast tree refers to the publisher of data in the tree.

Finally, existing multicast implementations have not paid sufficient attention to security concerns. Multicast schemes face some of the same security concerns as other applications, specifically ensuring confidentiality and authenticity of data transmitted over the network. However, the solutions that work in unicast do not apply directly to multicast. The publisher cannot negotiate a separate key for every subscriber in a system with any amount of scale. Using a shared symmetric key for the group raises the question of how to distribute such a key to new subscribers and revoke access from other subscribers.

Encryption alone cannot prevent side-channel or traffic analysis attacks. With many organizations loathe to give up access to their data (and some prevented from doing so by law), networking schemes to restrict the flow of data have a clear use case. Existing multicast solutions (and many networking solutions in general) do not take trust of the underlying network into account in routing. While assuming the underlying network to be untrustworthy has been the traditional network security assumption, relying on end-to-end encryption solutions alone is increasingly becoming untenable.

Our design of a new, edge-focused multicast is developed in conjunction with and targeted for the Global Data Plane (GDP) project, a publish/subscribe architecture for the Internet of Things (see chapter 2). The GDP seeks to shift the publish/subscribe model to favor the edge over data centers; as such, a new multicast system fitting this model was necessary.

Chapter 2

The Global Data Plane

We have developed Secure Content Distribution Trees in conjunction with the Global Data Plane (GDP) project. The SCDT concept is not exclusive to the GDP; however, the GDP is a case study of the applications benefited by SCDTs. We will reference the GDP throughout the remainder of this paper in order to demonstrate the broader infrastructure SCDTs are designed to operate within.

2.1 GDP Architecture for the Internet of Things

The Global Data Plane (GDP) is a data-centric abstraction focused around the distribution, preservation, and protection of information [60]. It supports the same application model as the cloud, while better matching the needs and characteristics of the IoT by utilizing heterogeneous computing platforms, such as small gateway devices, moderately powerful nodes in the environment and the cloud, in a distributed manner. As shown in Figure 2.1, the GDP interface provides a new "narrow waist" upon which applications are constructed. The basic foundation of the GDP is the secure, singlewriter log. Logs in the GDP are lightweight, durable, and they support multiple simultaneous readers—either through random access (pull-based) or subscription (push-based). Logs have no fixed location but rather are migrated as necessary to meet locality, privacy, or QoS needs of applications. Applications are built on top of the GDP by interconnecting log streams, rather than by addressing devices or services via IP. Each sensor or computational element of an IoT application has its own unique output log in the GDP and writes timestamped entries to this log. Actuators read from a unique input log. The GDP masks the heterogeneity of underlying communication paradigms, network/storage devices, and physical connections; and on top, it supports a wide variety of Common Access Application Program Interfaces (CAAPIs) for applications. We detail a few key design decisions below:

1. Single-writer time-series logs: For each IoT device or application component that generates data, this data is represented as a log where the owner has the sole write permission. This model is based on our observation that peripherals are physical devices in our

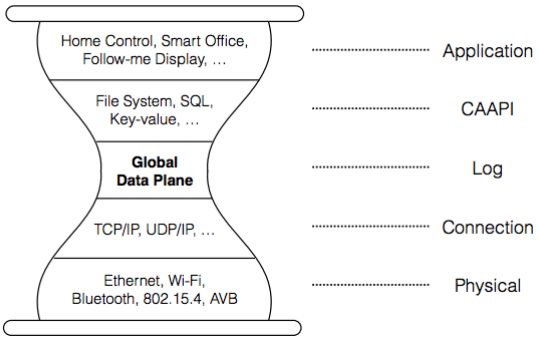

Figure 2.1: *The Global Data Plane (GDP) operates above the network level and offers Common Access APIs (CAAPIs) to applications rather than raw packet routing. We argue that this abstraction is more appropriate for both IoT applications and the cloud.*

environment. We assume that devices have cryptographic keys for signing and encryption. Logs are append-only; most data is readonly and can be securely replicated and validated through cryptographic hashes. For each log, our current design exposes append, read and subscribe APIs. The single-writer model allows the following properties:

- *Flexibility:* The log interface is minimum but complete. Aggregations of logs or CAAPIs (discussed below) can be built by composition. In part (a) of Figure 2.2, a new log is created by composing two existing ones and writing back to the GDP.

- *Access Control:* Since devices and services have associated public-key identities, each log has a single authorized writer. An append operation is permitted only when signed by the appropriate writer's key. For read operations, only those with an appropriate decryption key can decrypt the data, providing for a way to implement read-access control policies; a variety of more complex access control policies can be constructed through hierarchical key management or selected use of trusted environments.

- *Authenticity and integrity:* Since only signed append operations are allowed, accidental or malicious corruption of the log won't occur and substitution attacks are easily detected. A variety of traditional consistency problems are replaced with the simpler problem of finding the latest update.

- *Encryption:* We envision that all data written to the log is encrypted with the encryption key held by the writer. A single writer with a single encryption key simplifies the key management challenges.

- *Durability and replication:* In contrast to the cloud where users rely on whatever durability the cloud providers offer, our model enables the choice of the level of durability and geographic span of replication on a per log basis. The log model also simplifies replica consistency as previously mentioned.

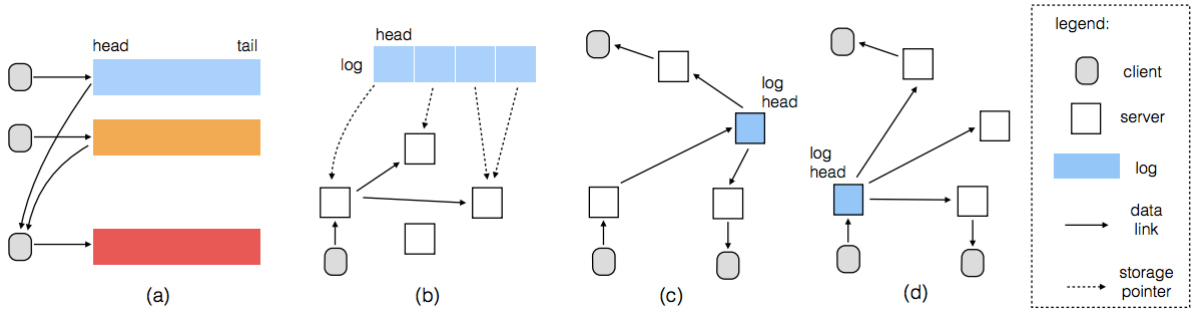

Figure 2.2: *The GDP design illustrated: (a) single-writer logs are appended to the head and compositions are achieved by subscription; (b) logs are split into chunks and stored in a distributed fashion; (c) overlay multicast trees are constructed when there are multiple subscribers; (d) location-independent routing enables log migration for optimizing performance.*

2. Location-independent Routing: Logs must be physically stored in the infrastructure. As previously discussed, the current reliance of IoT on cloud storage provides few guarantees about the placement, latency of access, or durability of information. Instead, to embrace heterogeneous platforms and support a variety of storage policies, the GDP employs location-independent routing in a large, 256-bit address space. To meet the goal of flexible placement, controllable replication and easy migration, packets are routed through an overlay network of routers running GDP software. GDP optimizes latency through log migration (see Figure 2.2(d)) and dynamic changes to the routing topology.

Logs are named with a 256-bit identifier which may be derived from a cryptographic hash of the owner's public key and metadata. Following a variety of placement and replication policies, the GDP places logs within the infrastructure and advertises the location of these logs to the underlying routing layer. Such placement and replication policies can optimize for latency, QoS, privacy, durability, and so forth. Internally, logs are further split into chunks, and each chunk can be distributed for durability [32] and performance [24] (see Figure 2.2(b)).

3. Pub/Sub and multicast tree: The publish/subscribe pattern has been shown to support a wide variety of fundamental communication services (for mobility, multicast, anycast [54]). This fits nicely with our log abstraction and can support building interactive applications. To alleviate the growth of sensor data bandwidth, when multiple subscribers exist, multicast trees can be built on top of the overlay network. We propose one such method in this thesis in chapter 4.

4. Common Access API (CAAPI): Although the singlewriter log abstraction shelters developers from low-level machine and communication primitives, many applications are likely to need more common APIs or data structures [6]. In fact, logs are sufficient to implement any convenient, mutable data storage repository. Thus, Figure 2.1 shows a CAAPI layer on top of the GDP. A CAAPI can provide key-value store, file system or database operations. Since logs serve as the ground truth, the benefit of consistency, durability, scalability and availability are carried over to CAAPIs for free. However CAAPIs may need to replay the logs if the service fails; in this case, checkpointing can be employed to avoid expensive

log replay.

The single-writer, append only log models sensor data more accurately; integrity and authentication by design provides better privacy and security; the distributed nature with multicast makes scalability possible; explicit separation of policy from mechanism enables better control on level of durability for end users; and finally, latency, bandwidth and QoS guarantees are enabled by the integration of the cloud and the local infrastructure.

2.2 Secure Content Distribution Trees in the GDP

The Global Data Plane Infrastructure and its pub/sub architecture offer a real-world case study for the application of Secure Content Distribution Trees (SCDTs), a networking protocol we will detail in the following chapters. SCDTs provide three main utilities to the GDP:

1. SCDTs provide the mechanism by which data is distributed to thousands or millions of geographically-disparate subscribers securely (and reliably, if necessary).
2. SCDTs provide the mechanism for distributing data among durable replicas.
3. SCDTs support fast distribution of data to local subscribers, enabling latency-dependent applications.

The GDP is designed to allow publishers to reach thousands or millions of subscribers. Often, the publisher and many (or all) of the subscribers are low-powered IoT devices. Those requirements necessitate a multicast scheme; a traditional client-server model is simply not scalable. Because the GDP is designed to support many different applications, it also requires reliable distribution of published data. Even ignoring reliable applications, durable replicas require reliability support.

The edge-focused nature of the GDP/SCDTs fills a need in pub/sub architectures not met by existing systems like Kafka, RabbitMQ, and ZeroMQ. All of those systems exist primarily in data centers; we have already discussed the drawbacks of this approach for the IoT in section 1.1. In addition, they don't account for the large number of overlapping data flows in a network involving many edge devices communicating with each other. This is a fundamentally different constraint than a relatively small number of publishers and subscribers communicating with a data center; in traditional applications, peer-to-peer communication is less common and client-server more prominent. Using GDP/SCDTs, the pub/sub system and the routing system are combined to increase efficiency, and can be scaled together. Deploying additional GDP routers on-site would generally improve performance, while with the older pub/sub systems scaling has to be done at the data center, which individual end-users may have limited or no control over.

One of the major, distinctive features of the IoT in general and the GDP in particular is the peer-to-peer nature of many applications. For instance, a street intersection might have

many "smart" devices that need to communicate amongst themselves with strong latency constraints (such as stop lights, street cameras, and car sensors) and with devices further away with weaker latency requirements (such as nearby intersections or a central control) in a city's traffic control system (see Figure 4.1). A traditional client-server model would require round trips to the cloud; one of the design goals of the GDP and SCDTs is to break out of this model to focus on supporting edge computing.

Chapter 3

Methodology

Our evaluation methods are based on simulations using ns-3 [44], a discrete-event network simulator written in C++. ns-3 allows us to simulate the entire Internet stack, from physical links up through link, network, and transport level protocols. Developing our applications for ns-3 closely models the development process for a full implementation. We developed SCDTs and CNR at the "application" level and "install" them onto our simulated nodes. Our applications are written comparably to real implementations and behave similarly. Each instance of our software is separated from each other instance; just like in a real network, they can only communicate over the network using sockets. There is no overarching control program coordinating the nodes in the simulation for us or otherwise simplifying coordination and communication among nodes. The interfaces to the ns-3 library, such as our simulated sockets, closely parallel the actual C++ socket interfaces, meaning that porting application code from an ns-3 simulation into an actual implementation would not require major re-architecting.

ns-3 allows us to manually specify a network topology, including the number of nodes, the connections between them, the protocols used at each layer, and a number of parameters such as bandwidth, network delay, and packet drop rate. We use ns-3 library implementations of the IP stack including UDP and TCP; our code sits on top of these, handling the application-level SCDT/CNR protocols. Both SCDTs and CNRs were coded in C++. The advantage of this approach is that our simulated SCDTs closely mirror a full standalone implementation; the disadvantage is that trying new algorithms is non-trivial, and dealing with both ns-3's and C++'s nuances makes rapid prototyping more difficult.

However, just because a given topology in ns-3 behaves as it would in the real world does not mean that the topologies selected necessarily reflect real-world deployments. To address this issue and avoid biasing our results by selecting hand-built topologies that favor our algorithms, we instead take advantage of BRITE [39] and BRITE integration in ns-3. BRITE, the Boston university Representative Internet Topology gEnerator, is a synthetic topology generation framework [11, 58] designed to create topologies that closely resemble the Internet in aspects including hierarchical structure and degree distribution. Using BRITE topologies for our simulations greatly strengthens our argument that SCDTs and CNR are

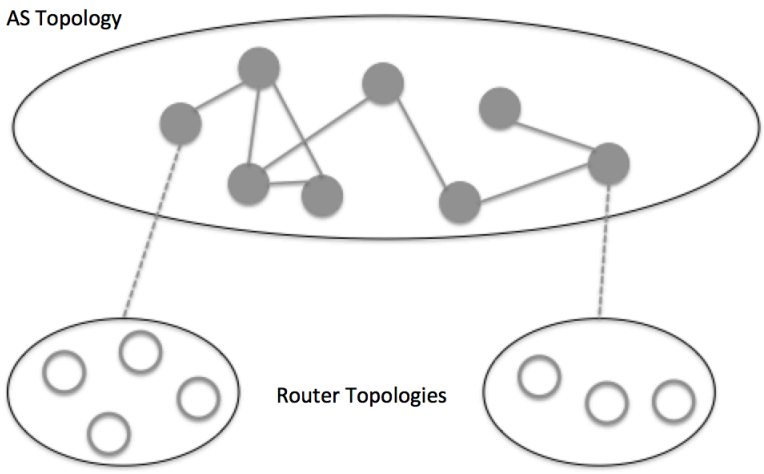

Figure 3.1: *A diagram of a top-down BRITE topology [38].*

practical in real-world deployments. BRITE offers a number of different models, which can be tuned to generate topologies with various numbers of ASes and nodes. Specifically, we configured BRITE to use a model composed of the the Barabási-Albert model [8] and the Waxman model [57] (see Figure 3.1), which are algorithms for generating network topologies. BRITE also offers many tunable parameters to tweak how the model performs; see Table 3.1 for more on BRITE parameters.

In each test run, we randomly select a subset of BRITE-generated leaf nodes to attach a node with our software "installed". In order to generate our results, we run our tests repeatedly in order to get results across many different BRITE-generated topologies and many different distributions of overlay-enabled routers. We then draw our results from the aggregate of this data, arguing that it is representative of an average use case of SCDTs and CNR on the Internet.

Parameter	Description
Flat Topology	
HS	Size of one side of the plane
LS	Size of one side of a high-level square
N	Number of nodes
Model	Model ID
alpha	Waxman-specific exponent
beta	Waxman-specific exponent
Node Placement	How nodes are placed in the plane
m	Number of links per new node
Growth Type	How nodes join the topology
BWdist	Bandwidth assignment to links
MaxBW	Max link bandwidth value
MinBW	Min link bandwidth value
Top-Down Hierarchical Topology	
Edge Connection	Method for interconnecting router topologies
Intra BWdist	Intra-domain bandwidth assignment distribution
Intra BWMax	Max bandwidth values
Intra BWMin	Min bandwidth values
Inter BWdist	Inter-domain bandwidth assignment distribution
Inter BWMax	Max bandwidth values for inter-domain links
Inter BWMin	Min bandwidth values for inter-domain links

Table 3.1: Selection of BRITE model parameters and their meanings [16].

Chapter 4

Secure Content Distribution Trees

4.1 Architecture

In this section, we outline the design of SCDTs. We make no claim that these solutions are optimal; rather, we chose these design patterns with implementation, deployment, and scalability in real-world environments in mind.

The fundamental SCDT structure is a multicast tree constructed out of an incrementally deployable overlay network [34] of routers and end-devices running SCDT software. The goal of this tree is to support a publish/subscribe model with a single, mobile publisher and a large number of subscribers. We argue for scalable tree-building mechanisms that can support reliable subscribers without bottlenecking the network. Our reliability design is based on well-researched negative acknowledgment schemes, with some changes to improve scalability for large numbers of IoT devices. Finally, we pay particular attention to securing SCDTs.

Heuristic, Adaptable Multicast

SCDTs rely on constructing overlay multicast trees to distribute data to large numbers of end-devices. We show an example in Figure 4.1, a "smart city" [59] deployment where street cameras must coordinate with traffic lights at the same intersection, traffic lights of neighboring intersections, and central servers far away. While Figure 4.1 only shows a half-dozen end devices, the analogy holds for thousands of end-devices across a smart city, including all traffic lights, crosswalks, and public transit systems. Building multicast trees on top of overlay networks has received significant study in the past [30, 12, 52]. However, these protocols often have questionable scalability, require significant work at the root of the tree, require IP multicast to be running underneath, or disregard latency or security concerns.

Building optimal multicast trees would require unrealistic amounts of overhead at the scale we are considering. Instead, SCDTs use heuristic methods. We describe the characteristics necessary for an IoT multicast tree building protocol with some similarity to that

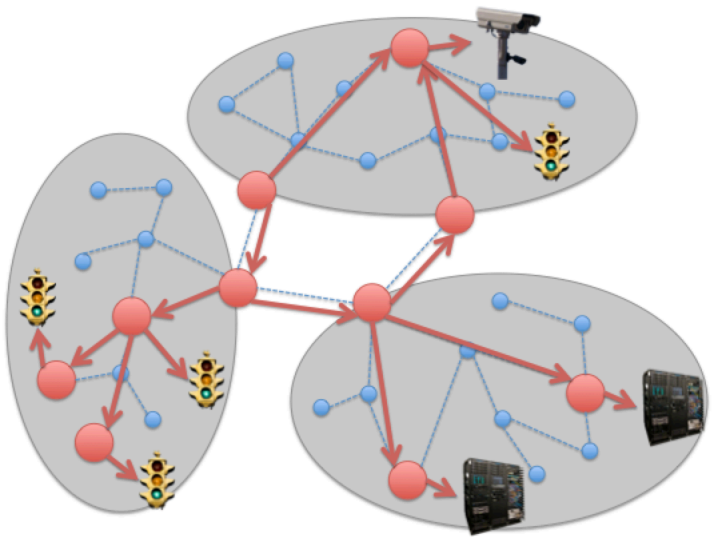

Figure 4.1: *An example SCDT running over a physical network with a traffic camera publishing data. The overlay network considers only the arrow links, which represent parent-child links. Legend: large nodes are running SCDT software; small nodes are not running SCDT software; arrow lines are overlay links; dashed lines are physical links; ovals are trusted domains.*

in [30], but with several critical modifications. Subscribers (and routers serving subscribers) attach to a nearby node in the SCDT and migrate up or down the tree to best satisfy its latency and bandwidth constraints.

Unlike many other solutions, we argue for a multicast model in which the publisher can move in the network and reattach to the SCDT in a different location. There are three key benefits to this model. First, it allows publishers to be mobile without having to regularly rebuild the entire tree. Second, it allows nearby nodes (e.g. those with real-time constraints) to receive data quickly, while still allowing more distant nodes to eventually receive data. Third, in the event of a break somewhere up the tree, nearby devices can still receive data quickly while the tree adapts. For instance, in Figure 4.1 a network fault which isolates the local intersection from the rest of the network should not cause interrupt service at the local intersection; instead, while it may reduce the effectiveness of nearby intersections which are no longer receiving the data, the local system should continue to function.

We also argue for a publisher/subscriber model in which there is only one publisher and arbitrarily many subscribers. This greatly simplifies tree construction and makes it easier to implement a mobile publisher.

End-devices themselves are not the true leaves of the SCDT. Rather, the routers these devices connect to should be considered the leaves of the SCDT. Since routers are often plugged-in and wired-in, this change in structure allows SCDTs to impose some processing load on the leaves without compromising low-power or low-resource devices. The leaves can determine for themselves when and how their heterogeneous end-devices should receive data.

Latency should not be disregarded. Network operators should be able to tune their

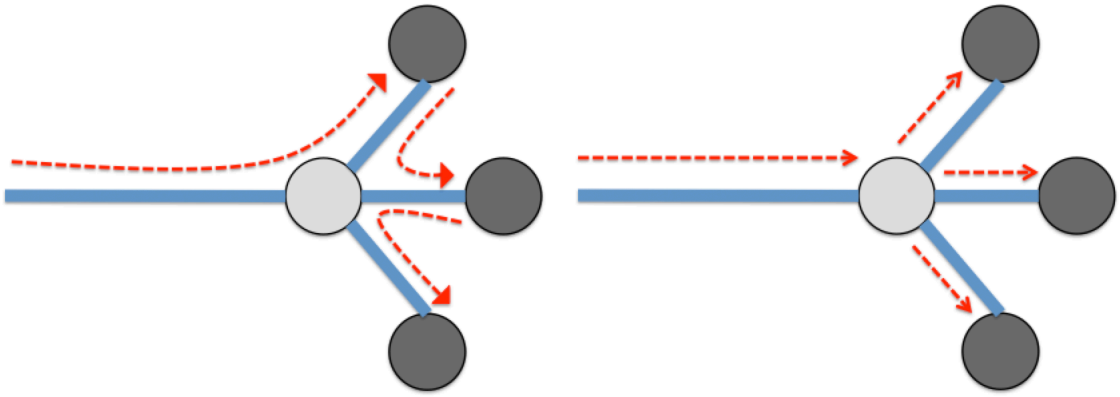

Figure 4.2: *Left: Overlay routing without forced participation, requiring unnecessary retransmission. Right: Overlay routing with forced participation. Children receive the packet faster, and the non-subscribing router handles fewer packets. Legend: light gray routers are not subscribing; dark gray routers are subscribing; solid lines are physical links; dashed lines show packet flow.*

devices for a worst case latency before optimizing for bandwidth. Latency may even be a more important factor than bandwidth when considering IoT devices which send data to subscribers relatively infrequently. Rather than trying to sample bandwidth, nodes should simply attach to a nearby parent. If they are unable to satisfy latency constraints or keep up with the data stream at their current location, they should migrate up/down the tree as necessary. Such a strategy avoids prematurely optimizing for bandwidth in trees which are only occasionally sending data, and helps to avoid unnecessarily long, snaking trees. In Figure 4.1, SCDTs provide low latency to the nearby traffic light which requires current data; meanwhile, devices further away with looser latency constraints will accrue greater bandwidth advantages.

We argue that the optimal metric for ensuring the above properties is *stretch*, introduced in section 1.3.

Rather than considering the Internet as a more or less randomly distributed graph of nodes, SCDTs consider the network as a series of interconnected, hierarchical domains of ownership. Domains are analogous to Autonomous Systems [26] and in many cases network operators might determine domain and AS boundaries to be the same. Border gateways of domains/ASes provide a natural choice for multicast points, and can help to service join requests and maintenance, reducing strain at higher levels of the SCDT. While this might provide a single point of failure (or relatively few points of failure), if the border gateways of an AS fail then there is no access to the broader Internet anyway. See section 4.2 for a discussion of the security aspects of domains.

We argue that overlay network routers that are not themselves subscribers to the data on a particular SCDT should be eligible to be drafted into service to increase tree efficiency. Figure 4.2 demonstrates a simple scenario in which router promotion would improve routing efficiency. By analyzing the way overlay links are constructed over the physical network, the SCDT can identify router promotions which would increase efficiency. The ability to

	SCDT	Overcast	Scribe
Locality Metric	Stretch	Bandwidth	Typically RTT or Hop Count
Re-Optimizes Routes	Yes	Yes	No
Mobile Publisher	Yes	No	No

Table 4.1: Comparison of various overlay multicast schemes.

promote unknown routers relies on the construction of trusted domains; only routers within the trusted domain should be eligible for promotion.

Secure Resolution Systems

A key construct of the SCDT system is the Secure Resolution System (SRS) provided by each domain. SRSes are roughly analogous to DNS, in that they are a hierarchical address resolution scheme. A new subscriber can contact its local SRS for information on border routers and attachments points for the desired SCDT. If the subscriber is the first in its domain, the SRS will point it to another SRS higher up the hierarchy.

Having all subscribers join the tree by contacting the root is cumbersome and slow. However, because SCDTs involve migrating between attachment points, new subscribers could theoretically use any existing node to join the tree. The closer the initially contacted node is to the ultimate attachment point, the more quickly the SCDT will converge to an optimal placement. The SRSes provide a simple way to find good attach points while allowing local administrators to configure the joining process if necessary. For instance, the local administrator might designate a single node as the domain attach point under the assumption that the domain will always keep that node running; or the administrator could configure a dynamic response based on the knowledge the SRS has about currently active nodes in the domain.

The implementation of the SRS is up to the domain administrator. A basic version on an SRS could simply be a database on a single server that contains border router information and pointers to other SRSes up the hierarchy; the IP address of this server could then be hard coded into all devices in the domain. A better version could be built using a Distributed Hash Table [53, 61].

Durability and Replication

While many IoT applications involve sending data for immediate consumption, it is also necessary to maintain a store of records somewhere in the network. The reasons for supporting such replicated, durable storage are threefold. First, end-devices which require reliable service need a final ground-truth to consult when data has already been completely purged from network caches. This is also the case for end-devices which go down and come back up later and need to consult data long forgotten by the routing infrastructure or even the

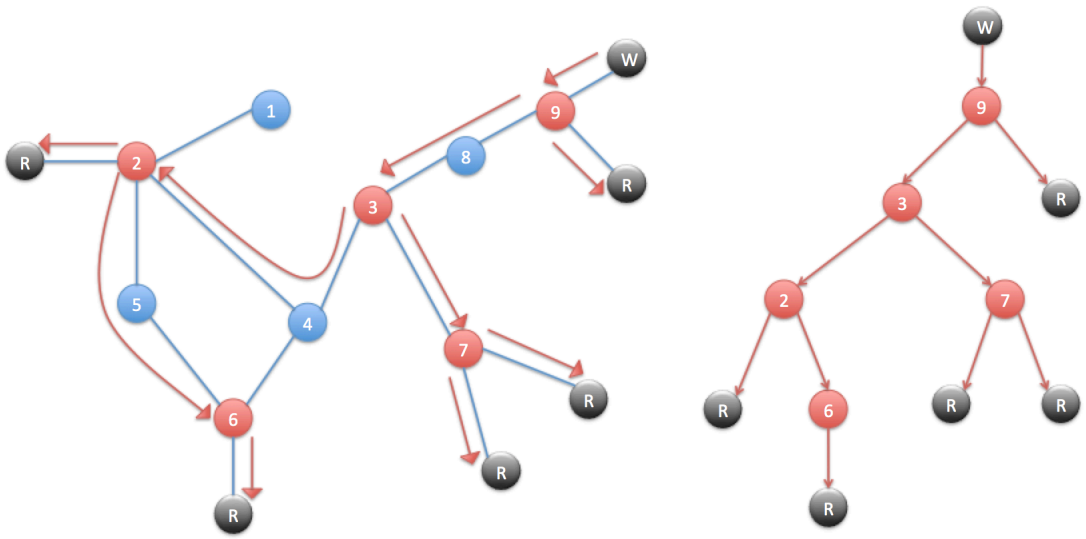

Figure 4.3: *Left: A physical network with SCDT nodes running on some machines. Right: The logical tree formed from that network. Legend: blue routers are not subscribing; red routers are subscribing; solid lines are physical links; dashed lines show packet flow.*

publisher itself. Second, many users will be interested in using analytics to derive insights from historical data. Third, some devices may only need certain pieces of data, and do not want to be fully joined to the SCDT.

Ideally, the SCDT can perform double duty, transporting data to interested end-devices as well as these ground-truth durable replicas.

Mobile Writer

We have designed SCDTs with a mobile publisher in mind. This means that a self-driving car or robot can reattach to the network elsewhere in the tree quickly. When the publisher rejoins the tree, routers can easily convert their previous parent connection to a child connection (since they're now receiving data from somewhere else). As the receivers continuously test their connections and migrate to better positions, the tree will eventually reform to a shape that better reflects the publisher's new position. The SCDT tree building protocol is extremely lightweight, allowing trees to quickly reform.

4.2 Security Implications

Denial of Service

Denial of service and amplification attacks are a major risk in SCDTs, since packets injected into the network will be rebroadcast. SCDTs are named using a secure namespace that allows many different SCDTs to coexist; specifically, they are named using a cryptographic

hash over the credentials of the creator of the tree, making trees attributable to owners and difficult to impersonate. Subscribers must present a certificate to join the SCDT, which can be verified at the join point, limiting the ability of attackers to attempt to spam the tree or eavesdrop on traffic. As discussed in section 4.1, we argue the use of trusted domains to restrict the open flow of data further reduces the risk of data exposure via side channel attacks. Additionally, the publisher signs data sent to the tree; packets without a valid signature will not be forwarded.

Broadcast Encryption Schemes

To achieve confidentiality efficiently, we argue SCDTs should utilize broadcast encryption techniques. Broadcast encryption [22] schemes, such as Subset Difference [43] or Layered Subset Difference [25], allow data to be efficiently encrypted and transmitted to a large number of receivers securely. In addition, the publisher can quickly revoke access from misbehaving subscribers by transmitting a limited number of update messages.

Trust Domains

Domains are a concept introduced by the GDP. A key feature of domains is trust (or lack thereof), primarily based on domain ownership, so that domains can serve as the boundaries within which data flows relatively freely. Organizations can then choose acceptable domains for their data to flow over, limiting data exposure risks. In order to transit further, developers must either trust other domains (e.g. their ISP) or establish highly secured channels between the border gateways of trusted domains. In many cases, ASes already fit the bill of trusted domains, so few modifications would be necessary to support SCDTs, but domains can also be much more specific, such as in the smart-home example. A simple example of the motivations for trust domains is a smart-home where input device commands must be sent to devices in the same home, but allowing those commands to leave the smart-home risks leaking important information, such as when a user comes and goes.

A more complicated network might involve a company's office infrastructure in one domain of trust, the company's factory in the next town in a second, and the company's ISP in a third. By grouping the underlying infrastructure into domains of trust, users can specify the flow of data in their networks, simplifying the process of securing data. This method helps to prevent side-channel attacks and other attempts to surreptitiously access encrypted data. For instance, previous research [45] has shown that analyzing the encrypted traffic of MapReduce jobs can reveal substantial amounts of the supposedly-secure data.

In our previous case, the company could restrict data flow based on their needs by specifying which domains they trust for which data. For example, door open/close notifications may only ever need to be routed to the on-site security staff, and could be restricted from flowing to the ISP, preventing a malicious actor outside the corporate network from learning the comings and goings of employees. The factory might enforce that commands sent

to robots on the factory floor cannot flow outside the building to prevent leakage of sensitive information about the manufacturing process. However, the company may mark its ISP's domain as trustworthy for high-level analytics data to move from the factory to the company's offices.

4.3 Evaluation

SCDTs primary differentiator from existing multicast schemes is the application of the stretch metric as the primary component for tree building. Trees are built by new nodes contacting the root, and then moving down the tree to the child that has the best stretch; the process continues until the joining node cannot move further down the tree without exceeding `MAX_STRETCH`. An example of a generated overlay topology is shown in Figure 4.4.

The goal of this metric is construct a tree that balances the need to fan out further down the tree for improve scalability with the real-time or near-real-time requirements of nearby IoT devices. Existing solutions do not take these real-time requirements into account. This solution is also more fault tolerant: partitions in the network, including losing connection to the broader Internet, will not prevent nodes on the same side of the partition as the publisher from continuing to receive content. Existing pub/sub architectures like RabbitMQ and Kafka are focused on the datacenter, and are not designed to account for device locality.

The SCDT is somewhat simplified for simulation purposes. Recall that the root node is the publisher in the tree and the first node to "join" the tree. Roughly, the simulation-version of the algorithm works as follows, assuming that `current_parent` is initially set to the root:

1. The joining node pings `current_parent` to determine its round-trip latency.
2. The joining node requests a list of `current_parent`'s children from `current_parent`.
3. The joining node pings each of these children to determine latency from itself to the child.
4. The joining node sends a request to each child requesting the child's latency to the root.
5. The joining node calculates *stretch* for each child as specified in section 1.3.
6. The joining node selects the child with the lowest stretch.
    a) If stretch is less than `MAX_STRETCH`, set this child as the `current_parent` and repeat this process.
    b) If stretch is greater than `MAX_STRETCH`, send an `ATTACH` request to `current_parent` and end the process.

In order to generate a stable topology for our simulations, our simulated SCDTs do not implement a re-optimization step. In a full implementation, SCDT nodes would periodically repeat the above process to account for changes in the network and tree structure after they joined The SCDT algorithm uses stretch as its primary metric rather than pure latency. This allows the SCDT to satisfy both of its primary goals: enabling real-time latency constraints and supporting a massively scalable pub/sub tree.

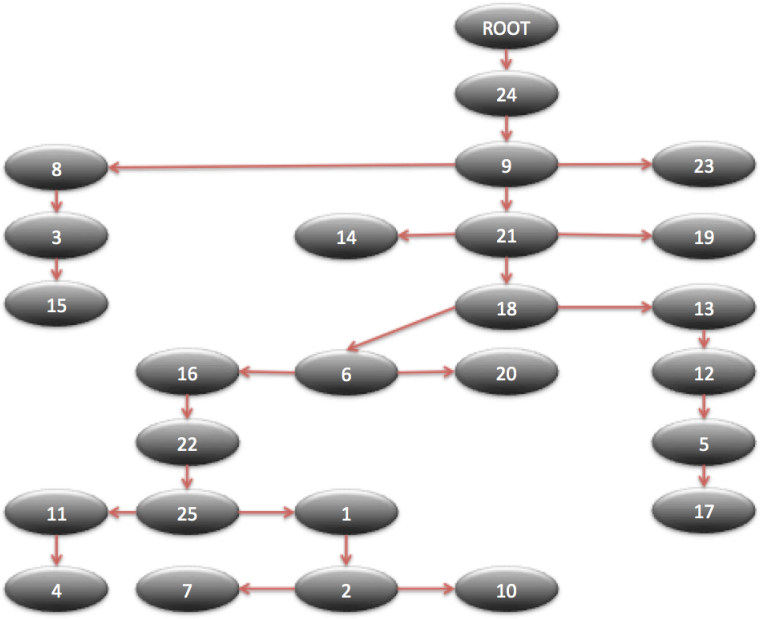

Figure 4.4: *The overlay multicast tree constructed out of 25 subscriber nodes and 1 publisher connected to a BRITE topology. Topologies vary from run to run due to the randomization of the BRITE topology and the link qualities.*

We simulated the impact of using SCDTs to distribute data. Our tests were constructed by generating a BRITE [39] topology consisting of two connected autonomous systems, and attaching SCDT nodes at the leaves of these ASes. BRITE topologies are regenerated for each run, helping to eliminate the effect of particular topologies skewing our results. See chapter 3 for more details. Link speeds are randomized to between 1 and 10 Mbps, and delays are randomized to between 1 and 50 ms.

Figure 4.5 demonstrates a test in which a packet is distributed to all SCDT subscribers and average latency is recorded. The latency subscribers encounter appears to increase linearly with the number of nodes in the tree, demonstrating the scale potential of SCDTs. However, we believe results in real-world deployments could scale even further. In our tests, nodes are randomly distributed; in reality, we would be more likely to see node clusters. In some of our tests, random distribution of nodes created a bottlenecking effect, with many nodes attaching to a single point; an intelligently constructed deployment could mitigate that issue.

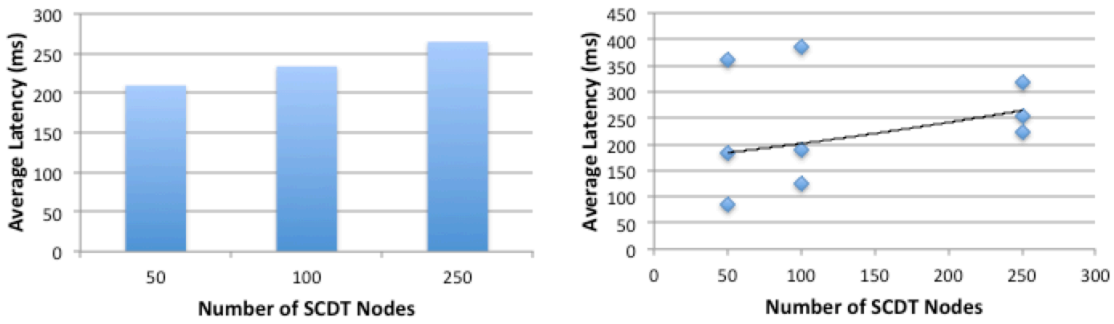

Figure 4.5: *Impact on latency of increasing the number of subscribers in the tree, with* `MAX_STRETCH` *set to 2. Left: The average latency of a single packet to subscribers over multiple runs. Right: The average latency of a single packet to subscribers, plotting individual runs and an exponential trend line.*

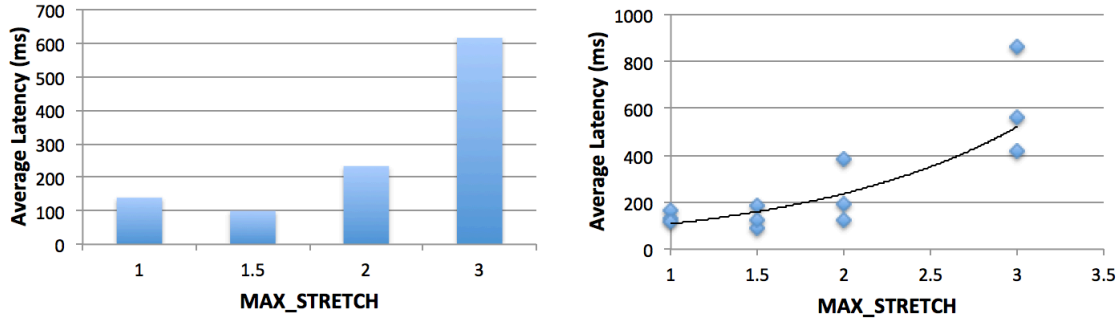

Figure 4.6: *Impact on latency of increasing the* `MAX_STRETCH` *parameter in the tree, with the number of subscribers fixed at 100. Left: The average latency of a single packet to subscribers over multiple runs. Right: The average latency of a single packet to subscribers, plotting individual runs and an exponential trend line.*

We also tested the stretch metric which is at the core of our algorithm. Figure Figure 4.6, shows the results of these tests. Based on this data, we believe the sweet spot for `MAX_STRETCH` tends to be between 1.5 and 2. Lower values had a tendency to severely limit node movement in the tree, resulting in nodes near the top of the tree with a large number of direct children. Our results indicate that as `MAX_STRETCH` is increased beyond 2, latency increases at an exponential rate. Examination of some of the constructed trees indicates an excessively high `MAX_STRETCH` leads to long, snaking trees with limited branching.

Figure 4.6 is also important for establishing the efficiency of SCDTs over other schemes. The case where `MAX_STRETCH` is set to one degenerates into a unicast relationship between the root and all children. This is obviously not a tenable situation as the trees continue to scale, but even at 100 nodes, SCDTs are about 30% faster than the basic unicast case with `MAX_STRETCH` is set to 1.5.

The results in Figure 4.6 are further reinforced by examining the average depth of nodes in SCDTs, show in Figure 4.7. As expected, tree depth tends to increase exponentially with

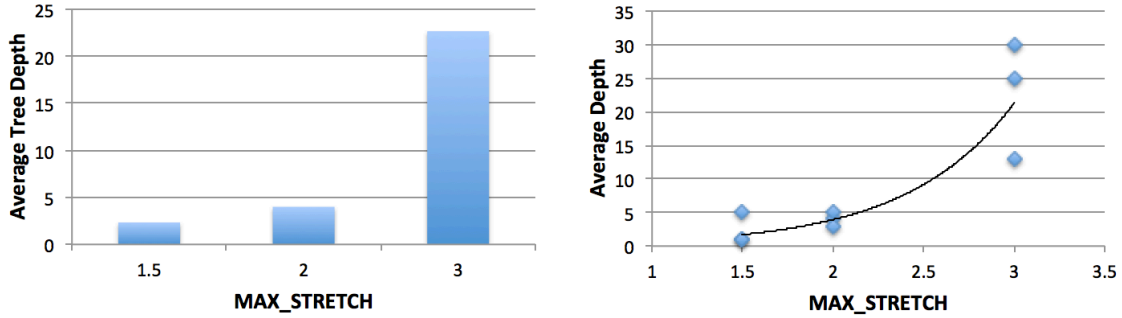

Figure 4.7: *Impact on tree depth of increasing the* `MAX_STRETCH` *parameter in the tree, with the number of subscribers fixed at 100. Left: The average depth of the tree over multiple runs. Right: The average depth of the tree plotting individual runs and an exponential trend line.*

`MAX_STRETCH`, leading to the aforementioned exponential increase in latency.

Our results show that tuning the `MAX_STRETCH` parameter for particular deployments will be critical. SCDTs, however, do allow a large degree of flexibility. The `MAX_STRETCH` value is set at the node level, not the network level, meaning that every node could have its own individually-tailored `MAX_STRETCH`.

This is an important property for real-world deployments. `MAX_STRETCH` could be set based on device priority; real-time applications could enforce a lower `MAX_STRETCH` while batch processing applications could settle for a much higher `MAX_STRETCH`.

It may also be important to tune this parameter based on where nodes (or clusters of nodes) are positioned in the network. Nodes located far from the root may counterintuitively require lower stretch values. These nodes will have a large latency when contacting the root, reducing sensitivity to placement in their local area (see section 1.3). This effect could also be offset by introducing intermediate join points in the tree, i.e. nodes that can serve as a proxy for the root, as discussed in section 4.1.

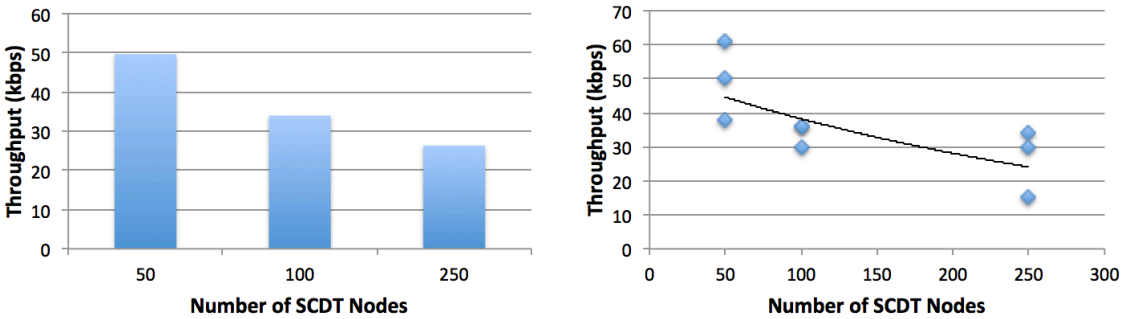

Figure 4.8: *Impact on throughput of increasing the number of subscribers in the tree. Sampled by sending 10KB with* `MAX_STRETCH` *fixed at 2. Left: The average throughput of the tree over multiple runs. Right: The average throughput of the tree plotting individual runs and an exponential trend line.*

Finally, we examine how SCDTs perform in terms of throughput. Figure 4.8 shows that average throughput decreases approximately linearly as the number of nodes increases. Note that these values should not be compared to the values in section 5.2 directly since each test was conducted under different parameters and configurations.

We also compared SCDTs throughput performance a naive tree building method fully described in section 5.2. While we defer precise description of the algorithm to chapter 5, in naive trees each node has a fixed fanout and chooses its children based on latency. In these tests, naive trees were built on top of a BRITE topology. Due to limitations in our simulations, we did not compare these at scale. However, we did compare the naive tree building method on a simulation containing 50 nodes. Throughput comparisons were found to be fairly comparable between the naive implementation and SCDTs at this size. Latency comparisons were also similar. While further scaling of these simulations is necessary to prove the effectiveness of SCDTs, we believe these results show that the concept has great promise.

Overall, SCDTs performed well in our tests. Both latency and throughput appear to scale well in our examinations. The most significant challenge is properly tuning the `MAX_STRETCH` parameter. However, we have many improvements in mind for SCDTs, which we discuss in section 6.1, which we believe will further improve SCDTs.

Chapter 5

Reliability in SCDTs

5.1 Cached Nack Reliability

Just like in traditional Internet services, IoT applications have a variety of reliability constraints. We propose that in a SCDT, the reliability should be constructed using negative acknowledgments ("nacks") from children. Previous research has already shown the negative acknowledgment scheme to be superior to regular acknowledgments in traditional network-level multicast trees [23, 52]; we argue that this principle extends to SCDTs. Unlike these previous schemes, SCDTs utilize caching at intermediate nodes. We argue that drafting intermediate routers as caches will improve scalability (by reducing the amount of traffic that must flow to the root) and improve partition tolerance (since retransmission could still occur even if the path to the root is lost). We call this scheme Cached Nack Reliability (CNR).

The SCDT forwards data unreliably to improve latency, but caches the data it forwards at each intermediate node. A traditional simple nacking scheme could use a similar method to that employed in TCP [47]: by examining an incrementally increasing sequence number associated with the SCDT included with every packet. Gaps observed in the sequence numbers of received packets indicate what data to nack. Periodic heartbeats sent by child nodes and acknowledged by parents keep sequence numbers updated even when data isn't frequently published. However, we argue for a more complex method: including a byte offset and packet length in the header of each packet. This method supports refragmentation of packets at intermediate points in the network.

Leaves can determine their reliability constraints for themselves, and send a nack for the missing data to their parent. If there is a cache hit, the data is retransmitted; if there is a miss, the nack is forwarded to the leaf's grandparent and so on, ultimately creating a hierarchy of caches. A slightly more sophisticated scheme could use timers with exponential backoff to reduce unnecessary and redundant nacking [23, 52]. Since we are arguing for a reliable system and the scheme presented so far relies on caches, there must ultimately be one or more places in the network where data is durably stored when caches all miss; see section 4.1 for more details.

Such a scheme provides a best of both worlds solution, minimizing latency while supporting packet retransmission. It breaks the traditional reliable vs unreliable (generally TCP vs UDP) trade off developers must choose between. By putting the impetus to nack packets on the leaf, rather than being completely reliable or unreliable, a leaf node could set a level of unreliability. For instance, a leaf node could choose to nack just enough packets to maintain a particular record reception rate.

5.2 Evaluation

We utilize a naive multicast tree building protocol to build the underlying multicast tree for our CNR tests. This allows us to evaluate CNR independently of SCDTs. In summary, the algorithm works as follows:

1. A joining node contacts the root and requests to join the tree.
2. The root pings the joining node to determine its round trip latency.
3. If the latency is substantially shorter than its existing children (or if the root has fewer children than `MAX_FANOUT`), the root adds the joining node as a direct child. If not, the root sends back a list of its children.
4. The joining node pings all of the children to determine which has the lowest latency.
5. The joining node repeats the process with the closest (determined by round trip time) child. The process is repeated until the joining node finds a parent that will accept it.

We evaluate CNR in comparison to another baseline algorithm. Our naive reliability algorithm simply uses TCP links between every parent and child, essentially creating a series of point-to-point TCP links. Our results are predicated on comparing this naive baseline to CNR.

Using point-to-point TCP presents a number of issues in actual deployments. One is the risk of bottlenecking the entire tree due to one bad link, where a router's buffer becomes full and must drop incoming packets because it cannot push out data to one of its children fast enough. Another issue is the high computational cost of setting up TCP links, which is non-ideal if the tree is continuously shifting and re-optimizing. While we don't advocate using point-to-point TCP in multicast trees, it is a useful contrast point because it represent a fairly direct comparison to reliability in the unicast space.

Preliminary simulation results for CNR are generated on a star topology with the root in the center, using the naive trees (not SCDTs). Because of the limited fanout of our trees, we nonetheless still build meaningful multicast trees on this topology. See chapter 3 for more information about our simulation environment. Link speeds are randomized to between 1 and 100 Mbps, and delays are randomized to between 1 and 100 ms.

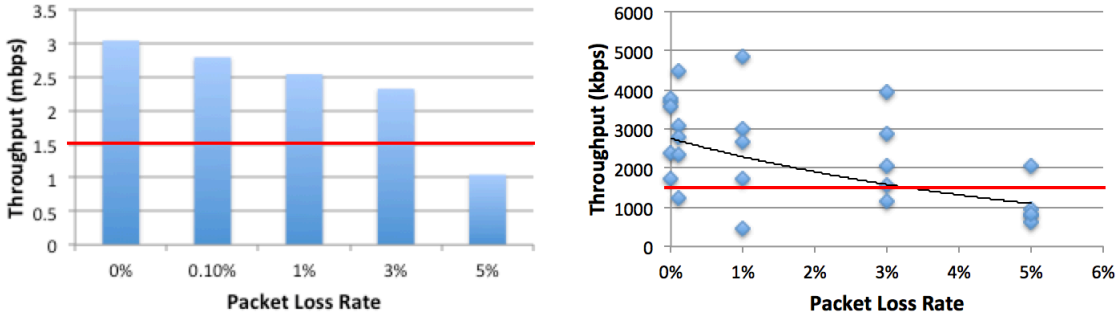

Figure 5.1: *Impact on throughput of packet loss rate using CNR. Sampled by sending 100KB to a tree containing 50 subscribers and a* `MAX_FANOUT` *of 4. The red line represents the average throughput of TCP over several runs with no packet loss. Left: The average throughput of CNR over multiple runs. Right: The average throughput of CNR plotting individual runs and an exponential trend line.*

Our results are summarized in Figure 5.1. We sent 100KB of data to all the subscribing nodes in our multicast tree, and measured the throughput. We then introduced packet drops into the network, and measured the throughput of CNR when packets were randomly dropped 0.1%, 1%, 3%, and 5% of the time. The data suggests that CNR performs well in the face of fairly substantial packet loss, with fairly minor performance degradation until packet losses grow above 3%, after which performance reductions become more substantial.

What is particularly interesting, however, is how much better CNR performed compared to hop-to-hop TCP links, even with no packet loss. Our data shows CNR generating substantially greater throughput than hop-to-hop TCP even when CNR is experiencing 3% packet loss and TCP is experiencing none; the breakeven point is somewhere between 3% and 5% packet loss. While this is admittedly hardly the use case TCP was designed for, we believe this demonstrates the superiority of our approach and of nacking in multicast applications in general. We attribute the poor performance of hop-to-hop TCP to the overhead imposed by the TCP protocol and the loss of end-to-end efficiency [50] TCP generally relies on. As discussed in section 4.1, however, using many end-to-end TCP connections simply does not scale with the number of subscribers we are considering.

We believe this result will only improve with increased scale. Our test described in Figure 5.1 considers only 50 nodes and a `MAX_FANOUT` of 4, meaning that a packet would traverse at most 3 overlay hops to reach its destination. In a larger network (or, in some cases, in an SCDT), the number of hops would be greatly increased. In a system without caching, this would impose substantially greater round trip times. However, we did not test the effect of varying cache sizes and the impact of cache misses on CNR performance.

Chapter 6

Wrapping Up

6.1 Future Work & Lessons Learned

We've shown SCDTs to be a viable networking structure for the Internet of Things. However, many of our discussed optimizations were not included in our simulations. Implementing any number of these would improve the performance of SCDTs even further.

For instance, in our simulated nacking scheme we always return data in fixed size blocks. However, fragmentation in actual networks could lead to nacks which request byte ranges that don't conform to block boundaries. For example, we might cache 100 byte blocks on the parent containing bytes 1-100, 101-200, and 201-300, but the child might nack bytes 50-150. Currently, our simulation would respond with 1-200; an optimized implementation could return only bytes 50-150.

Our simulations of tree building do not allow nodes to join at intermediate points in the network, an optimization that will ultimately be critical for scalability. Instead, our simulated nodes always join at the root. In addition, we do not include the re-optimization step in our simulations, which would allow nodes to shift their position in the tree after joining.

We do not measure network stress in our simulations. Measuring stress requires modifying the network stack on all simulated nodes to monitor network-level traffic to observe the movement of individual packets over every link. While it is fully possible to do this, we do not include it in our simulations at this time.

On the other hand, we have not simulated the impact of a mobile publisher. While the effect of an actively moving publisher will likely have a negative impact on the performance of the algorithm, we do not believe that this effect will be overly damaging. The performance impact of SCDT security mechanisms was also not evaluated in these simulations, though the algorithms and techniques we selected were specifically chosen for their applicability and low-overhead in applications comparable to SCDTs.

While we believe we have proved the utility of SCDTs, further work to implement SCDTs and test them in real-world deployments is certainly necessary. We hope to take many of the

design principles of SCDTs and implement them in the Global Data Plane (see section 2.2), a rapidly developing infrastructure of the Internet of Things.

6.2 Conclusion

We have presented the design and architecture of Secure Content Distribution Trees, a networking protocol targeting the Internet of Things. We argue that existing networking protocols do not address the scale of the Internet of Things or the real-time and locality aspects of many applications. Our architecture is designed to address these dual goals above all else. Simulations indicate that SCDTs are a promising avenue for future edge networking research.

We have also presented an improved multicast reliability scheme, Cached Nack Reliability. While we introduce this algorithm in the context of SCDTs, it is a viable approach to reliability for any multicast scheme, including both IP multicast and overlay multicast.

The Internet of Things presents enormous challenges and opportunities. Increasingly pushing computing systems to the edge of the network has already begun to upend the traditional networking infrastructure, which prioritized mainly-downstream traffic from a relatively small number of data centers to largely-independent terminals and devices. SCDTs can help to change that paradigm, pushing more traffic to the edge and de-emphasizing the cloud.

Bibliography

[1] A. Adams, J. Nicholas, and W. Siadak. *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*. RFC 3973. RFC Editor, Jan. 2005.

[2] B. Adamson et al. *NACK-Oriented Reliable Multicast (NORM) Transport Protocol*. RFC 5740. RFC Editor, Nov. 2009.

[3] Sanjay Aiyagari et al. *Advanced Message Queuing Protocol*. Tech. rep. Dec. 2006. URL: https://www.rabbitmq.com/resources/specs/amqp0-9.pdf.

[4] *AMQP 1.0 Discussion Paper: Broker Behavior*. Tech. rep. June 2010. URL: https://www.rabbitmq.com/wp-uploads/2010/11/amqp-broker-prototype.pdf.

[5] Emmanuel Baccelli et al. *RIOT: One OS to Rule Them All in the IoT*. Tech. rep. [Research Report] RR-8176, INRIA 2012. hal-00768685v3 . Dec. 2012.

[6] Mahesh Balakrishnan et al. "Tango: Distributed Data Structures over a Shared Log". In: Nov. 2013. URL: https://www.microsoft.com/en-us/research/publication/tango-distributed-data-structures-over-a-shared-log/.

[7] A. Ballardie. *Core Based Trees (CBT) Multicast Routing Architecture*. RFC 2201. RFC Editor, Sept. 1997.

[8] Albert-László Barabási and Réka Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509–512. ISSN: 0036-8075. DOI: 10.1126/science.286.5439.509. eprint: http://science.sciencemag.org/content/286/5439/509.full.pdf. URL: http://science.sciencemag.org/content/286/5439/509.

[9] "BeagleBone". In: (). www.beagleboard.org/bone.

[10] Iljitsch Van Beijnum. "IPv6 celebrates its 20th birthday by reaching 10 percent deployment". In: (Jan. 2016). https://arstechnica.com/information-technology/2016/01/ipv6-celebrates-its-20th-birthday-by-reaching-10-percent-deployment/.

[11] Ken Calvert, Matt Doar, and Ellen W. Zegura. "Modeling Internet Topology". In: June 1997.

[12] M. Castro et al. "Scribe: a large-scale and decentralized application-level multicast infrastructure". In: *IEEE Journal on Selected Areas in Communications* 20.8 (Oct. 2002), pp. 1489–1499. ISSN: 0733-8716. DOI: 10.1109/JSAC.2002.803069.

[13] Yang-hua Chu et al. "A Case for End System Multicast". In: *IEEE Journal on Selected Areas in Communications* 20.8 (Oct. 2002), pp. 1456–1471. ISSN: 0733-8716. DOI: 10.1109/JSAC.2002.803066.

[14] Nigel Cory. "Cross-Border Data Flows: Where Are the Barriers, and What Do They Cost?" In: (2017). https://bit.ly/2DU4D6y.

[15] Steve Deering. *Host extensions for IP multicasting*. STD 5. http://www.rfc-editor.org/rfc/rfc1112.txt. RFC Editor, Aug. 1989. URL: http://www.rfc-editor.org/rfc/rfc1112.txt.

[16] *Design and Implementation of BRITE*. Tech. rep. Apr. 2001. URL: https://www.cs.bu.edu/brite/user_manual/BritePaper.html.

[17] C. Diot et al. "Deployment issues for the IP multicast service and architecture". In: *IEEE Network* 14.1 (Jan. 2000), pp. 78–88. ISSN: 0890-8044. DOI: 10.1109/65.819174.

[18] A. Dunkels, B. Gronvall, and T. Voigt. "Contiki - a lightweight and flexible operating system for tiny networked sensors". In: *29th Annual IEEE International Conference on Local Computer Networks*. Nov. 2004, pp. 455–462. DOI: 10.1109/LCN.2004.38.

[19] D. Estrin et al. *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*. RFC 2362. RFC Editor, June 1998.

[20] Patrick Th. Eugster et al. "The Many Faces of Publish/Subscribe". In: *ACM Comput. Surv.* 35.2 (June 2003), pp. 114–131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: http://doi.acm.org/10.1145/857076.857078.

[21] B. Fenner et al. *Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")*. RFC 4605. http://www.rfc-editor.org/rfc/rfc4605.txt. RFC Editor, Aug. 2006. URL: http://www.rfc-editor.org/rfc/rfc4605.txt.

[22] Amos Fiat and Moni Naor. "Broadcast Encryption". In: *Advances in Cryptology — CRYPTO' 93*. Ed. by Douglas R. Stinson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 480–491. ISBN: 978-3-540-48329-8.

[23] S. Floyd et al. "A reliable multicast framework for light-weight sessions and application level framing". In: *IEEE/ACM Transactions on Networking* 5.6 (Dec. 1997), pp. 784–803. ISSN: 1063-6692. DOI: 10.1109/90.650139.

[24] Trinabh Gupta et al. "Bolt: Data Management for Connected Homes". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 243–256. ISBN: 978-1-931971-09-6. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/gupta.

[25] Dani Halevy and Adi Shamir. "The LSD Broadcast Encryption Scheme". In: *Proceedings of the 22Nd Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '02. London, UK, UK: Springer-Verlag, 2002, pp. 47–60. ISBN: 3-540-44050-X. URL: http://dl.acm.org/citation.cfm?id=646767.704291.

[26] J. Hawkinson and T. Bates. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*. BCP 6. RFC Editor, Mar. 1996.

[27] Pieter Hintjens. *ZeroMQ. Messaging for Many Applications*. O'Reilly Media, 2013.

[28] H. Holbrook, B. Cain, and B. Haberman. *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*. RFC 4604. `http://www.rfc-editor.org/rfc/rfc4604.txt`. RFC Editor, Aug. 2006. URL: `http://www.rfc-editor.org/rfc/rfc4604.txt`.

[29] *Information technology – Telecommunications and information exchange between systems – Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)*. Standard. Geneva, CH: International Organization for Standardization, Nov. 2002.

[30] John Jannotti et al. "Overcast: reliable multicasting with on overlay network". In: *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation-Volume 4*. USENIX Association. 2000, p. 14.

[31] Jay Kreps, Neha Narkhede, Jun Rao, et al. "Kafka: A Distributed Messaging System for Log Processing". In: *Proceedings of the NetDB*. 2011, pp. 1–7.

[32] John Kubiatowicz et al. "OceanStore: An Architecture for Global-scale Persistent Storage". In: *SIGPLAN Not.* 35.11 (Nov. 2000), pp. 190–201. ISSN: 0362-1340. DOI: `10.1145/356989.357007`. URL: `http://doi.acm.org/10.1145/356989.357007`.

[33] Philip Levis et al. "TinyOS: An Operating System for Sensor Networks". In: *Ambient Intelligence*. Vol. 00. Jan. 2005, pp. 115–148. ISBN: 978-3-540-23867-6.

[34] Eng Keong Lua et al. "A survey and comparison of peer-to-peer overlay network schemes". In: *IEEE Communications Surveys Tutorials* 7.2 (Feb. 2005), pp. 72–93. ISSN: 1553-877X. DOI: `10.1109/COMST.2005.1610546`.

[35] Gary Scott Malkin. *RIP Version 2*. STD 56. `http://www.rfc-editor.org/rfc/rfc2453.txt`. RFC Editor, Nov. 1998. URL: `http://www.rfc-editor.org/rfc/rfc2453.txt`.

[36] James Manyika et al. "The Internet of Things: Mapping the Value Beyond the Hype". In: (2015). `http://bit.ly/2gyPezB`.

[37] M. McBride, J. Meylor, and D. Meyer. *Multicast Source Discovery Protocol (MSDP) Deployment Scenarios*. BCP 121. RFC Editor, Aug. 2006.

[38] Alberto Medina et al. *BRITE: Universal Topology Generation from a User's Perspective*. Tech. rep. BU-CS-TR-2001-003. Apr. 2001.

[39] A. Medina et al. "BRITE: an approach to universal topology generation". In: *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 2001, pp. 346–353. DOI: `10.1109/MASCOT.2001.948886`.

[40] Bill Morelli et al. *IoT Trend Watch 2018*. IHS Markit, 2018.

[41] J. Moy. *Multicast Extensions to OSPF*. RFC 1584. RFC Editor, Mar. 1994.

[42] John Moy. *OSPF Version 2*. STD 54. http://www.rfc-editor.org/rfc/rfc2328.txt. RFC Editor, Apr. 1998. URL: http://www.rfc-editor.org/rfc/rfc2328.txt.

[43] Dalit Naor, Moni Naor, and Jeff Lotspiech. "Revocation and Tracing Schemes for Stateless Receivers". In: *Advances in Cryptology — CRYPTO 2001*. Ed. by Joe Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 41–62. ISBN: 978-3-540-44647-7.

[44] "ns-3". In: (). https://www.nsnam.org/.

[45] Olga Ohrimenko et al. "Observing and Preventing Leakage in MapReduce". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: ACM, 2015, pp. 1570–1581. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813695. URL: http://doi.acm.org/10.1145/2810103.2813695.

[46] J. Polastre, R. Szewczyk, and D. Culler. "Telos: enabling ultra-low power wireless research". In: *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* Jan. 2005, pp. 364–369. DOI: 10.1109/IPSN.2005.1440950.

[47] Jon Postel. *Transmission Control Protocol*. STD 7. http://www.rfc-editor.org/rfc/rfc793.txt. RFC Editor, Sept. 1981. URL: http://www.rfc-editor.org/rfc/rfc793.txt.

[48] "Raspberry Pi". In: (). www.raspberrypi.org.

[49] Antony Rowstron and Peter Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: *Middleware 2001*. Ed. by Rachid Guerraoui. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 329–350. ISBN: 978-3-540-45518-9.

[50] J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-to-end Arguments in System Design". In: *ACM Trans. Comput. Syst.* 2.4 (Nov. 1984), pp. 277–288. ISSN: 0734-2071. DOI: 10.1145/357401.357402. URL: http://doi.acm.org/10.1145/357401.357402.

[51] C. Shields and J. J. Garcia-Luna-Aceves. "The ordered core based tree protocol". In: *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*. Vol. 2. Apr. 1997, 884–891 vol.2. DOI: 10.1109/INFCOM.1997.644571.

[52] T. Speakman et al. *PGM Reliable Transport Protocol Specification*. RFC 3208. http://www.rfc-editor.org/rfc/rfc3208.txt. RFC Editor, Dec. 2001. URL: http://www.rfc-editor.org/rfc/rfc3208.txt.

[53] Ion Stoica et al. "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications". In: *IEEE/ACM Trans. Netw.* 11.1 (Feb. 2003), pp. 17–32. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.808407. URL: http://dx.doi.org/10.1109/TNET.2002.808407.

[54] Ion Stoica et al. "Internet Indirection Infrastructure". In: *IEEE/ACM Trans. Netw.* 12.2 (Apr. 2004), pp. 205–218. ISSN: 1063-6692. DOI: 10.1109/TNET.2004.826279. URL: http://dx.doi.org/10.1109/TNET.2004.826279.

[55] D. Thaler. *Border Gateway Multicast Protocol (BGMP): Protocol Specification*. RFC 3913. RFC Editor, Sept. 2004.

[56] D. Waitzman, C. Partridge, and S.E. Deering. *Distance Vector Multicast Routing Protocol*. RFC 1075. RFC Editor, Nov. 1988.

[57] B. M. Waxman. "Routing of multipoint connections". In: *IEEE Journal on Selected Areas in Communications* 6.9 (Dec. 1988), pp. 1617–1622. ISSN: 0733-8716. DOI: 10.1109/49.12889.

[58] J Winick et al. "INET: An autonomous system (AS) level Internet Topology generator, Ver. 3.0". In: *University of Michigan, Technical Report CSE-TR-456-02* (2002).

[59] A. Zanella et al. "Internet of Things for Smart Cities". In: *IEEE Internet of Things Journal* 1.1 (Feb. 2014), pp. 22–32. ISSN: 2327-4662. DOI: 10.1109/JIOT.2014.2306328.

[60] Ben Zhang et al. "The Cloud is Not Enough: Saving IoT from the Cloud". In: *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA: USENIX Association, 2015. URL: https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/zhang.

[61] Ben Yanbin Zhao, John Kubiatowicz, Anthony D Joseph, et al. "Tapestry: An infrastructure for fault-tolerant wide-area location and routing". In: (2001).