1-2014

From Map to Dist: the Evolution of a Large-Scale Wlan Monitoring System

Keren Tan
Dartmouth College

Chris McDonald
The University of Western Australia

Bennet Vance
Dartmouth College

Chrisil Arackaparambil
Dartmouth College

Sergey Bratus
Dartmouth College

See next page for additional authors

Recommended Citation

Keren Tan, Chris McDonald, Bennet Vance, Chrisil Arackaparambil, Sergey Bratus, and David Kotz. From MAP to DIST: the evolution of a large-scale WLAN monitoring system. In IEEE Transactions on Mobile Computing, January 2014. 10.1109/TMC.2012.237

Authors
Keren Tan, Chris McDonald, Bennet Vance, Chrisil Arackaparambil, Sergey Bratus, and David Kotz

From MAP to DIST: The Evolution of a Large-Scale WLAN Monitoring System

Keren Tan, Chris McDonald, Bennet Vance, Chrisil Arackaparambil, Sergey Bratus, and David Kotz, *Fellow*, *IEEE*

Abstract—The edge of the Internet is increasingly becoming wireless. Therefore, monitoring the wireless edge is important to understanding the security and performance aspects of the Internet experience. We designed and implemented a large-scale WLAN monitoring system, the Dartmouth Internet security testbed (DIST), at Dartmouth College. It is equipped with distributed arrays of "sniffers" that cover 210 diverse campus locations and more than 5,000 users. In this paper, we describe our approach, designs, and solutions for addressing the technical challenges that have resulted from efficiency, scalability, security, and management perspectives. We also present extensive evaluation results on a production network, and summarize the lessons learned.

Index Terms—Network measurement, optimization, wireless network, 802.11, WLAN, security, scalability

1 Introduction

The edge of the Internet is increasingly becoming wireless. Therefore, monitoring the wireless edge is important to understanding the security and performance aspects of the Internet experience. This is especially necessary for enterprise-wide wireless local-area networks (WLANs) as organizations increasingly depend on WLANs for mission-critical tasks. For the past decade, our research team at Dartmouth College has continuously devoted effort to developing new technologies, software tools and systems to measure large-scale WLANs [1], [2], [3], [4] and to carry out extensive security analysis on these networks [5], [6], [7].

Monitoring a WLAN, especially a large-scale one, is a difficult undertaking. Our previous work [1], [2] and many other WLAN measurement studies [8], [9] have monitored the wired side of access points (APs) in infrastructure WLANs using SNMP, syslog, and packet sniffing. These techniques monitor the traffic that has been bridged from the wireless edge to the wired core of a network. The views offered by such techniques are often incomplete because they only characterize *how* the monitored WLAN and its users behave, and have provided little insight about *why* the network and its users behave in such a manner [10]. We were among the first to explore the feasibility of using distributed arrays of air monitors (AMs) to passively monitor the IEEE 802.11 link layer and higher layers [6]. Since then, via passive monitoring, many WLAN traces have been collected in conference events, a building floor and even a whole building; several distributed WLAN monitoring systems, such as DAIR [11], Jigsaw [10], MAP [6], and AirLab [12] have been successfully built and used for WLAN security and management research.

This paper describes our experience gained during the design, implementation, and management of a distributed large-scale WLAN monitoring system, the Dartmouth Internet security testbed (DIST). As one of the largest WLAN monitoring systems, DIST is equipped with 420 radio interfaces on 210 AMs, and covers 11 buildings and more than 5,000 users. In the MAP project [6], the predecessor of the DIST project, we implemented a building-wide WLAN monitoring system. However, when we attempted to scale the deployment to a campus, the MAP system could no longer meet the required levels of efficiency, scalability, security, and manageability. Our new monitoring system was designed to address these challenges. We made three major contributions in designing and building DIST.

Saluki. A high-performance Wi-Fi sniffing system. Compared to our previous implementation and to other available sniffing programs, Saluki has many advantages:

1. its small memory and computation footprint makes it suitable for a resource-constrained Linux platform, such as those in commercial Wi-Fi access points;
2. all traffic between the sniffer and the back-end server is secured using 128-bit encryption;
3. the frame-capture rate has increased more than threefold with minimal frame loss;
4. under the same frame-capture rate, the traffic load on the backbone network is reduced to only 30 percent of that in our previous implementation.

DISTSANI. An online network trace sanitization and distribution program. It receives the network trace captured by Saluki, sanitizes several fields in the frame/packet headers, and distributes the sanitized network trace to different destinations simultaneously. The implemented

- *K. Tan, B. Vance, C. Arackaparambil, S. Bratus, and D. Kotz are with the Institute for Security, Technology, and Society, Dartmouth College, 7 Maynard Street, Hanover, NH 03755. E-mail: keren@outlook.com, {bennet, cja, sergey, kotz}@cs.dartmouth.edu.*
- *C. McDonald is with the School of Computer Science and Software Engineering, The University of Western Australia, 35 Stirling Highway, Crawley, WA 6009, Australia. E-mail: chris.mcdonald@uwa.edu.au.*

Manuscript received 12 Dec. 2011; revised 27 July 2012; accepted 25 Oct. 2012; published online 20 Nov. 2012.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2011-12-0670.
Digital Object Identifier no. 10.1109/TMC.2012.237.

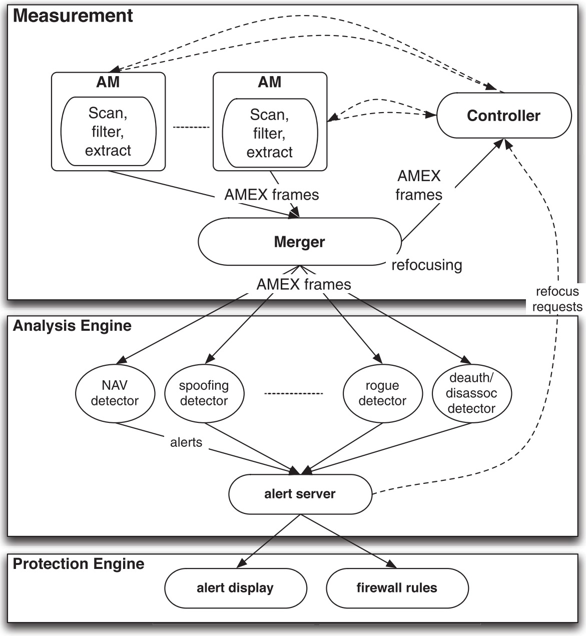


Fig. 1. MAP architecture.

sanitization process is highly efficient, processing up to three million addresses per second.

MAPmaker. A tool for configuring, launching, monitoring, and terminating an experiment. A running experiment consists of interacting processes distributed across many hosts, including both servers and AMs. MAPmaker pushes master executables for these processes to the hosts that need them, remotely starts them, and keeps track of their process id numbers (pids) both for monitoring purposes and so that the experiment can be shut down in an orderly fashion. MAPmaker runs multiple independent experiments concurrently without interference among them.

2 Background

MAP [6] aimed to build a security-focused WLAN monitoring system while DIST has broader goals. The architecture of the MAP system is shown in Fig. 1. Between 2005 and 2007, we deployed 20 Aruba AP70s [13] flashed with OpenWrt Linux [14] in the computer science department building at Dartmouth College. We used Aruba AP70s as air monitors (AMs) to capture the MAC-layer headers of wireless frames, then extract and forward the desired frame features to the merger process, which creates a unified stream on a coherent timeline. The analysis engine includes plug-in detectors that analyze the traffic, producing alerts to the protection system and feedback to the measurement system. The adoption of Aruba AP70s enabled our work on IEEE 802.11b/g networks, but not on 802.11n, which was only standardized 4 years later.

MAP includes several advanced features for WLAN measurement: AM feature extraction, AM channel sampling and refocusing, and multisource trace merging. AM feature extraction is designed to reduce the volume of forwarded traffic. It works as a user-configurable filter that extracts user-requested information from each captured frame/packet, and only forwards the extracted information to the server in a frame format named AMEX. AM channel sampling and refocusing are two complementary strategies to deal with the multichannel-monitoring problem in WLAN. The unlicensed bands used for IEEE 802.11 networks have multiple channels, and a single-radio AM can only listen to one channel at any time. One could attach multiple radios to one device, or place multiple single-radio devices at one location. Either way, the hardware required is bulky or prohibitively expensive. The AMs in MAP monitor multiple channels by periodically assigning the radio to each channel, dynamically adjusting and coordinating the schedule to maximize capture [4], [15]. This technique is named channel sampling, as it collects only a sample of the frames passing through all the channels. MAP supports multiple sampling strategies, including *equal-time sampling*, which spends the same amount of time on each channel, *proportional sampling*, which spends more time on the busiest channels, and *coordinated sampling*, which minimizes channel overlap between neighboring AMs. However, AM channel sampling will inevitably lose information because the AM only visits each channel for a limited time. To compensate for this loss, MAP allows the analysis components to dynamically *refocus* the measurement system after observing some user-defined suspicious behaviors, by gathering more frames from a client, AP, or region, or by extending the set of features collected about the traffic of interest [3]. In the event of an ongoing network attack, the higher fidelity stream of frames may allow MAP to confirm the attack or locate the attacker. We refer interested readers to our previous work [6] for more thorough information about MAP and its comparison to Jigsaw [10], DAIR [11], DOMINO [16], and Wit [17].

For DIST, we aimed to cover a large portion of the Dartmouth College campus with AMs. Dartmouth College was among the first universities in the world to provide campus-wide WLAN coverage. In 2001, more than 500 Cisco 350 APs were installed, to provide campus-wide IEEE 802.11b service. In 2006, this WLAN migrated to an Aruba Networks solution that provides IEEE 802.11a/b/g services simultaneously. More than 1,300 Aruba AP70 access points were installed to cover 1.8 square miles of campus populated by over 6,000 students and 2,500 faculty and staff.

3 Challenges

We faced many challenges when designing and implementing DIST: performance and scalability, security and privacy, and management and monitoring. We address each group in turn.

3.1 Performance and Scalability

Table 1 compares the scale of MAP and Jigsaw [10] to DIST. It can be seen that DIST's scale is much larger than that of either MAP or Jigsaw. DIST deploys over 10 times as many AMs, covering 11 buildings in approximately a 1:1 ratio to the access points that provide 802.11a/b/g connectivity in those buildings. The density and placement of the DIST AMs are intended to enable capture of traffic both from the access points they shadow and from those access points' clients. In a typical 24-hour span, the

TABLE 1
Deployment Scale Comparison of MAP, Jigsaw, and DIST

	MAP	Jigsaw	DIST
Deployed AMs	20	96	210
Radio interfaces	40	192	420
Covered buildings	1	1	11
Covered users	≈ 100	≥ 1000	≥ 5000

210 AMs capture and forward more than 500 gigabytes of IEEE 802.11 MAC layer headers.

We use the Aruba AP70 [13] flashed with OpenWrt Linux [14] as our AM's hardware platform. The advantage of the Aruba AP70 is that it fully complies with IEEE 802.3af standard for Power over Ethernet (simplifying installation), provides diverse interfaces (USB, serial, and Ethernet), and has a compatible appearance to other devices in our deployment environment. Because the Aruba AP70 was originally designed to be a commercial AP instead of a wireless AM, its processing capability is limited: 266-MHz MIPS 4Kc CPU, 28-MB RAM, and 8-MB flash memory storage. Indeed, just to put it in context, modern cellphones have more memory and CPU power. In our previous MAP project, we developed a sniffing system named dingo [3], [4]; it supports several advanced features, such as channel sampling, data aggregation, dynamic filtering, and refocusing. However, dingo's performance deteriorates quickly when dealing with high-volume traffic. The significantly increased deployment scale and dingo's limited performance compelled us to design and implement a new high-performance Wi-Fi sniffing program for DIST.

Because DIST is distributed across campus, the whole system works in a client/server mode: the sniffing programs run on the remote AMs, capturing and forwarding traffic to our servers via the campus backbone network. The expected high volume of captured data also drove us to consider its impact on the backbone network. DIST servers are located in the computer science department. Since these servers share a 1-Gbps link with more than 200 other machines in the department, more than 500 gigabytes per day through this link would negatively affect other machines' network performance. To efficiently use the available bandwidth, and to alleviate the pressure on the shared medium, effective data aggregation and compression are features essential to DIST.

3.2 Security and Privacy

Since DIST is monitoring Dartmouth's production WLAN, used daily by thousands of students and staff, the collected traces contain sensitive information related to the activities of humans and their devices; identifiers such as MAC addresses may identify individuals and their location.

We conducted an extensive security analysis and built detailed threat models of the DIST system in our previous work [18]. In this paper, we focus on two kinds of threats that may jeopardize the data flow transmitted inside the DIST system. First, an adversary may intercept the traffic between the AM and server. Second, an adversary may have access to the server that stores captured traces. To protect against the first threat, we require all data exchanged between AMs and the servers, including both captured traces and control messages, to be encrypted to ensure data confidentiality. As a further step, we implemented a hash-based message authentication code (HMAC) to ensure both the data integrity and the authenticity of all data exchanged between AMs and the servers. To protect against the second threat, we require all received data to be sanitized before being written to persistent storage (such as hard drives) or sent to data subscribers (including the merger and users' analysis components, except for real-time attack detection). As our AMs capture and forward only MAC-layer headers to the servers, there is no higher layer data, such as TCP/IP headers and payloads, to sanitize. Fig. 2 shows the types of information flows inside DIST. We adhered to a strict guideline when transferring and storing captured data: *if it is not encrypted, then it must be sanitized.*

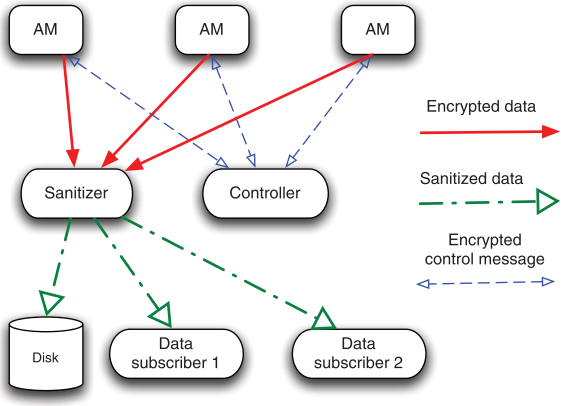


Fig. 2. DIST information flows.

3.3 Management and Monitoring

It is a challenging task to *seamlessly* and *continuously* configure, run and monitor DIST's diverse devices and software components—its 210 AMs together with the sanitizers, mergers, and other processes running on the four DIST servers.

Although there are a small number of infrastructure monitoring tools available, such as Nagios [19] and Cacti [20], they do not fit the needs of DIST for several reasons. First, most of the devices in DIST are Aruba AP70s, which are resource-constrained embedded Linux devices. Installing and running the software that is used to monitor more traditional Linux infrastructure is not economical and exhausts the limited resources on such embedded devices. Second, to seamlessly and continuously run the whole system, DIST needs not only the monitoring functionality but also "automatic-configuring" and "self-resetting" functionalities. As implied by its name, DIST is a large-scale general-purpose testbed that should be able to run whatever experiments DIST can support. The duration of these experiments may range from several hours to several days, weeks, or even months. We need the "automatic-configuring" functionality so that we can create diverse experiment configurations intuitively and efficiently. We need the "self-resetting" functionality so that in the event of some exception, the system can reset, reconfigure, and rerun the affected devices and applications.

TABLE 2
Comparison of Passive Network Sniffing Programs

		tcpdump	Wireshark	Kismet	dingo	Jigdump	Saluki
Features	wired/wireless network	Both	Both	Wireless	Wireless	Wireless	Both
	client/server mode	No	No	Yes	Yes	Yes	Yes
	data aggregation	No	No	No	Yes	Yes	Yes
	data compression	No	No	No	No	Yes	Yes
	data encryption	No	No	No	Yes	No	Yes
	data authenticity	No	No	No	No	No	Yes
	data integrity	No	No	No	No	No	Yes
	Wi-Fi channel sampling	No	No	Yes	Yes	No	Yes

4 Approach

In this section, we detail how DIST addresses the above performance and scalability, security and privacy, management and monitoring challenges.

4.1 Saluki

While Saluki shares many of the same features as other passive network sniffing software tools, its design has been driven by our past experience and the special needs of the DIST project (performance, scalability, security, and privacy). Table 2 provides a detailed comparison between Saluki and other well-known passive network sniffing programs, such as tcpdump [21], wireshark [22], Kismet [23], dingo [6], and Jigdump [10]. In this table, only Saluki provides the complete feature set to address DIST's efficiency, scalability, and security challenges. It is worth noting that Jigdump is also a highly efficient sniffing program but, because of its dependence on Atheros chipset and a specific old version of MadWifi driver [24], it lacks the portability of other sniffing programs. We describe Saluki in detail in a workshop paper [25] and dissertation [26] and summarize it here.

4.1.1 Capture Interface

We use a raw socket with PACKET_MMAP enabled as the capture interface. The raw socket lets us bypass the protocol stacks (the link layer and above) inside the Linux kernel, and the memory mapping provides for efficient communication between kernel space and user space. This interface avoids inefficiencies introduced by abstractions in the libpcap library.

In the Linux kernel, PACKET_MMAP is specifically designed to facilitate the network traffic capturing task. Without this socket interface, capturing each network packet requires a system call. PACKET_MMAP implements a configurable circular buffer between the user and the kernel space—capturing a packet in the user space becomes a simple read operation on the shared circular buffer [27]. This interface proved highly efficient on the AP70s. In one test (simply capturing frames and counting, nothing else), this interface was able to capture 7,063 frames per second (fps) with 25-35 percent CPU usage and 3.3 percent frame loss. As a comparison, tcpdump with libpcap 0.9.8 under the same traffic load froze the AMs.

4.1.2 Data Aggregation

Saluki uses UDP packets to forward the captured traffic back to our central servers. We observed that if we pack only one frame in each UDP packet, the 100-Mbps Ethernet connection on the Aruba AP70 could not keep up when there was a high volume of wireless traffic. We measured the maximum throughput under different UDP datagram sizes; for example, 10-byte UDP datagrams achieved only 45-KBps throughput whereas 1,500-byte datagrams achieved 5,327 KBps (where KBps = 1,000 bytes per second). Small UDP packets degrade the Ethernet throughput greatly. Given that small frames, like a 14-byte ACK frame, are widely used in the IEEE 802.11 MAC layer, it is much more efficient to aggregate multiple frames and then send them as a "combo" frame. A DIST combo frame has two sections: the header contains metainformation about the combo frame, and the data section holds multiple captured frames. When a new frame is captured, Saluki appends the frame size and the frame content to the DIST combo frame's data section.

It is worth noting that there is a tradeoff between the size of the combo frame and the frame-receipt delay at the server side. While a bigger combo frame will use the Ethernet connection more efficiently, bigger is not always better, especially for time-critical applications, like wireless-network intrusion detection. For this reason, we defined two adjustable criteria to decide when a combo frame should be sent: when the payload size of a combo frame exceeds a size threshold, or when the time difference between the first enclosed IEEE 802.11 frame and the current system clock exceeds a time threshold. In our current implementation, we set these two parameters to 14 KB and 1 s, respectively.

4.1.3 Data Compression

The DIST combo frame increases Saluki's network efficiency, but we need to do better to more efficiently use the shared 1-Gbps backbone Ethernet bandwidth, so we compress a combo frame before sending it. Given the Aruba AP70's limited processing power, instead of pursuing the maximum compression ratio, we aimed to find a lossless compressor that has a good balance between processing speed and compression ratio.

After some background study, we focused on two variants of the Lempel-Ziv (LZ) compression method [28]: QuickLZ [29] and FastLZ [30]. Compared to the standard LZ compressor, these two variants trade compression ratio in favor of speed. It is important to note that a compressor's performance (compression ratio and speed) may vary when dealing with different data. We chose QuickLZ because it had a more consistent performance on our captured network data. In our experiments, a 14-KB combo frame was compressed to 2.8-3.6 KB by QuickLZ. The use of compressed combo frames saved more than 70 percent of the load on the backbone network, compared with sending individual uncompressed frame headers in each UDP packet.

4.1.4 Data Encryption

As a basic security measure to protect the privacy of the network users whose traffic we capture, we employ a fast stream cipher to encrypt all traffic between each AM and the central servers. The stream cipher employs a 128-bit IV to significantly mitigate the opportunity for commonly appearing Radiotap and IEEE 802.11 headers to be enumerated by an adversary. Although our communication between AM and servers is carried in UDP/IP datagrams, we have consciously chosen not to employ a standard datagram transport layer security (DTLS) protocol, as there is no reverse channel between servers and AMs to carry acknowledgments. To support rapid frame capture and collection in an environment where many frames are knowingly missed on channels not being monitored, we are willing to tolerate lost UDP/IP datagrams.

We evaluated all stream ciphers from the eSTREAM project [31] and the SNOW 2.0 cipher [32]. The best two ciphers were Rabbit and SNOW 2.0, which have been accepted as ISO standard stream ciphers (ISO/IEC 18033-4). Both of them support 128-bit encryption and are much faster than RC4 and AES in counter mode [31].

We evaluated an assembly language implementation of the Rabbit cipher optimized for the MIPS 4Kc processor, whereas SNOW 2.0 is implemented in the C language and was not specifically optimized for this processor. Since our goal was to transmit the protected data most efficiently, we tried the ciphers both without compression and in combination with compression. We observed the following:

1. For stream ciphers, Rabbit emerged as a winner on the Aruba AP70, surpassing SNOW 2.0. When executing 5,000 loops on 14-KB data, Rabbit took 5.33-5.55 s, whereas SNOW 2.0 took 7.42-7.73 s.
2. Adding compression decreases the total processing time, because there were fewer bytes to encrypt. In effect, compression was computationally "free."

Securely transmitting 5,000 14-KB combo frames (each combo frame may contain tens to hundreds of captured Radiotap and IEEE 802.11 frames) to a DIST server took 6.2-6.4 s, which encompassed two operations: encryption and UDP forwarding. The load on the network averaged 14 KB per combo frame. If we compressed these combo frames first, however, handling them took *less* time, namely, 5.3-5.4 s for *three* operations: compression + encryption + UDP forwarding. The required network bandwidth was also reduced by more than 70 percent (from 14 KB per combo frame to 2.8-3.4 KB per combo frame). This result illustrates that efficient compression not only saves network bandwidth, but also reduces CPU time needed for encryption and UDP forwarding. If needed, we could set the size of the uncompressed DIST combo frame to be larger than 14 KB. Although this change might improve the network throughput, it would increase the delay at the server.

4.1.5 Data Authenticity and Integrity

Encryption provides data confidentiality but does not ensure data authenticity and integrity. To achieve a higher level of security, we integrated an optional HMAC-SHA256 component into Saluki's communication.

HMAC-SHA256 is one type of HMAC that uses a SHA-256 cryptographic hash function [33], [34]. The input to HMAC has two parts: the data to be processed, and a secret key. HMAC's strength depends on the size of the secret key. Currently we use a 128-bit secret key, but this key can be lengthened to 256 bits to improve HMAC's security against a brute-force attack. The output of HMAC-SHA256 is a 256-bit message authentication code (MAC). For the collected network traces transmitted from AMs to servers, this 256-bit MAC is generated by Saluki running on AMs and verified by the receiver program, DISTSANI, running on servers.

To accurately quantify HMAC-SHA256's performance impact, we developed a benchmark program derived from Olivier Gay's HMAC-SHA2 implementation [34]. The program was executed 50,000 times for a variety of data block sizes between 1,000 and 5,000 bytes. The HMAC-SHA256 implementation had a constant overhead of 0.1 ms per block, and a near-linear throughout of 1.33 MBps. From this, we estimate that when the frame capture rate is 5,500 fps, capture length is 192 bytes for each frame, and an uncompressed DIST combo frame is 14,000 bytes, HMAC-SHA256's CPU usage on AP70s will be 5.24, 7.53, and 9.82 percent for 2,000, 3,000, and 4,000 bytes of compressed DIST combo frames, respectively.

4.1.6 Multithreading

So far we have introduced four core components of the Saluki sniffing program: capture interface, data aggregation, compression, and encryption. The final important task is to assemble them efficiently. Each of these components is relatively self-contained and can work independently from other components. For example, capturing frames from the Wi-Fi interface and forwarding DIST combo frames via Ethernet are I/O-intensive operations, while data compression and encryption are CPU-intensive operations. This observation inspired us to fit these components into a multithreading pipeline. We experimented with several combinations of component and thread placement, and Fig. 3 shows our final and optimal configuration.

From Fig. 3, we see that Saluki has three threads. Threads 1 and 2 undertake data capturing, processing, and forwarding. Thread 3 is the control thread, managing scheduling and channel-hopping tasks (as in dingo [3], [4]). Two ring buffers are used in this program. The top ring buffer is responsible for mapping the captured frames from kernel space to user space and is emptied by Thread 1. The second ring buffer connects Thread 1 with Thread 2. From the perspective of multithreaded programming, the communication through these two ring buffers follows the classic writer/reader programming model.

Instead of putting compression, encryption and UDP forwarding all in Thread 2, we had planned to divide them between two threads: compression and (optional) HMAC computation in one thread; encryption and UDP

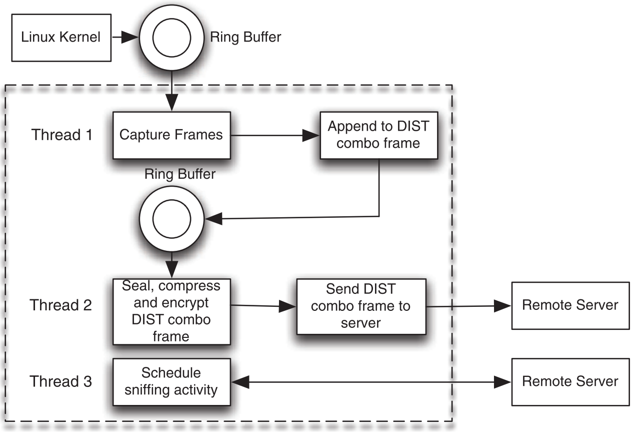

Linux Kernel
Ring Buffer
Thread 1
Capture Frames
Append to DIST combo frame
Ring Buffer
Thread 2
Seal, compress and encrypt DIST combo frame
Send DIST combo frame to server
Remote Server
Thread 3
Schedule sniffing activity
Remote Server

Fig. 3. The Saluki architecture.

forwarding in the other. In the test run, however, we observed that Thread 1 was the bottleneck here: its CPU usage was about 1.5-2 times that of Thread 2 even when HMAC was turned on in Thread 2. Due to this observation, we did not split Thread 2 further.

Fig. 4 shows the data flow inside Saluki when all the features (data aggregation, encryption, compression and the optional HMAC computation) are turned on.

4.2 DISTSANI

As noted above, we take the position that if our captured data is not encrypted then it must be sanitized. In DIST, the transformation from encrypted data to sanitized data happens inside DISTSANI. As the "sanitizer" in Fig. 2, DISTSANI is responsible for parsing the received DIST combo frames, sanitizing each 802.11 frame, and distributing sanitized 802.11 frames to their correct destinations. These three functionalities are implemented as three components in DISTSANI.

4.2.1 Parsing and Distributing Components

The parsing component receives the DIST combo frames sent by Saluki and mirrors the operation of Saluki: decrypt a DIST combo frame using the Rabbit algorithm, decompress it using the QuickLZ algorithm, and split this combo frame into individual 802.11 frames. The distributing component packs the processed 802.11 frames using pcap format and outputs them to different destinations, according to user specifications—either to a trace file on local hard drives or to a live UDP stream forwarded to data subscribers. For the same input traffic, different users may receive different outputs from DISTSANI because they can choose different ways to sanitize the same captured data.

4.2.2 Sanitizing Component

DIST policy requires that all data available to data subscribers (except for real-time attack detection) or stored to persistent storage must be sanitized. This dictates that the DIST sanitization process must be *online* and be fast enough to keep up with the data capturing speed; otherwise, much captured data will be lost. Although many sanitization algorithms have been proposed, few of them have an online version capable of performing at line speed [35], [36]. We developed a network trace sanitization library called *libdistsanitize* that incorporates different sanitization methods. Of note, our sanitization scheme only sanitizes MAC and SSID addresses [18], and not any other addresses or identifiers that appear in frame payloads, such IP addresses, TCP ports, or email addresses. The primary reason for this is that DIST only captures and stores MAC layer headers together with physical-layer interface details, such as channel frequency and signal strength. Moreover, most contemporary wireless networks encrypt everything above the MAC layer, making sanitization both impossible and unnecessary.

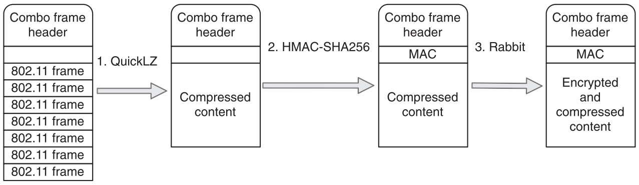


Fig. 4. The data flow inside Saluki.

Each MAC address consists of 48 bits. Our sanitization scheme preserves the two most significant bits, which indicate if an address is used for unicast or multicast communication and if the address is globally or locally administered. These two bits are simply copied from each raw address to its sanitized equivalent. The other 46 bits are handled as follows.

We employ an SHA-2-based cryptographic hash function to produce a stream of 256-bit hash values, each providing eight 32-bit pseudorandom integers. A per-trace sanitization key seeds the random number generation. These random numbers are employed to generate two large mapping tables—one providing a one-to-one mapping between each address's 22-bit IEEE-assigned organizationally unique identifier (OUI) field and a distinct 22-bit value, and the other similarly mapping the 24-bit vendor supplied SERIAL number field to a distinct 24-bit value. Thus, for each invocation of DIST, each possible 48-bit raw MAC address has a corresponding, unique, 48-bit sanitized address. Depending on the experiments being undertaken, sanitization of either field is optional and selected at runtime. When sanitizing only the 22-bit OUI field, the field indexes the OUI table producing the 22-bit sanitized OUI field, and the 24-bit SERIAL field is copied, verbatim, to produce the output SERIAL field. When sanitizing a full MAC address, we consider the 46 significant bits as 23 odd bits and 23 even bits. We use the 22 least significant odd bits to form a 22-bit OUI index, and the 23 even bits and the remaining odd bit to form a 24-bit SERIAL index. These indices locate the sanitized values in the mapping tables.

Our sanitization is a deterministic one-to-one process, so that each MAC address is consistently mapped to a distinct MAC address within the same trace. The generation of the mapping tables needs to be performed only once per experiment, has a fixed memory footprint of 80 MB, and executes in 4.3 s on our 3-GHz servers. This memory footprint, and the need to securely distribute and then discard the sanitization key, preclude the sanitization being performed on the AMs. As the process of sanitizing a MAC address simply involves extracting its OUI and SERIAL fields, two function calls and two array accesses, the process is extremely fast. On our 3-GHz servers, "pass-through" sanitization (i.e., a no-op) takes $0.078\ \mu s$ per address, sanitization of just the OUI or SERIAL fields takes $0.281\ \mu s$ per address, and sanitization of both fields takes $0.382\ \mu s$ per address (each time averaged over 10 million addresses).

4.3 MAPmaker

MAPmaker is our tool for configuring, launching, monitoring, and terminating an experiment. MAPmaker also generates and distributes the keys required for Saluki's cipher. MAPmaker runs on the experiment's master host and uses secure channels (ssh) to control the DIST activity on all the DIST servers and AMs.[1]

A running experiment consists of interacting processes distributed across many hosts, including both servers and AMs. Rather than keep these processes' executables permanently resident on all the hosts—which would necessitate pushing any updates to all affected hosts when new executables were built—we instead maintain master executables on only one master host.

At experiment start-up time, MAPmaker pushes the required executables to the hosts that need them, starts them as background processes, furnishes them with keys, and keeps track of their process id numbers (pids). While the experiment proceeds, MAPmaker periodically checks that the registered pids are still alive and, if not, takes remedial action (restarting a failed executable, restarting the entire experiment, or shutting it down, depending on the nature of the failure). At the conclusion of the experiment, MAPmaker kills all participating processes on all hosts in an orderly fashion.

The master executables are neither signed nor verified. Because we push all files to AMs and servers through an ssh connection, with firewalls to limit inbound access to the DIST server, we consider this precaution unnecessary. We have considered, but not implemented, mechanisms by which our AMs could receive and verify encrypted executables pushed from the DIST master server. However, as the AMs are not in a physically controlled environment, and the chosen hardware does not include trusted hardware able to verify executables, we can have no assurance that the AMs are not executing untrusted software. We have also considered the dual challenge of having the DIST master server remotely verify the AMs' running software through a challenge-response protocol, but similar difficulties exist.

4.3.1 Approach to Configuration

Part of MAPmaker's mission is to distribute configuration information to the participating processes. For example, processes communicating through sockets need to agree on the port numbers to use, and need to know each other's IP addresses. The various processes also need to refer to paths in the experiment's timestamped file tree so that they can cooperatively collect a coherent set of traces, as well as metadata such as log files. A MAPmaker configuration uses symbolic names to refer to the experiment's port numbers, IP addresses, file paths, and a variety of other parameters. At experiment start-up, MAPmaker instantiates these symbolic names as concrete values in the command-line parameters and helper files for individual processes. In this way, the same symbolic configuration information can be shared by multiple instances of the same executable; for example, all 210 of our AMs typically use the same helper file, which is instantiated with appropriate variations at the different AMs. MAPmaker's systematic use of symbolic parameters and its tracking of pids also make it possible to run concurrent, independent instances of nearly identical experiment configurations without any interference among them.

4.3.2 Implementation

MAPmaker is implemented as a collection of Python scripts. An individual DIST configuration must provide MAPmaker enough information to infer the concrete values for the configuration's parameters. Rather than defining parameters through a file containing name-value pairs, which would limit the structural complexity of the parameters, or through a structured document format such as XML, which would have required us to write code to translate the XML definitions into variable values that could be used by the Python scripts, we let parameters be defined directly as Python variables in a small Python program that serves as a MAPmaker configuration and works in concert with the MAPmaker scripts. This approach gives the parameter definitions the full expressive power of the Python language.

The parameter definitions are grouped into a hierarchy of Python classes, with each leaf class representing a DIST-system component that MAPmaker must be able to start, monitor, and stop. For example, our configurations include a class `Merger` that provides the definitions that MAPmaker needs to manage a configuration's mergers. We typically divide the 210 AMs into seven merging regions, each of which is served by its own merger instance. This approach allows mergers to run in parallel on servers physically close to areas of heavy traffic collection. The `Merger` class contains six class-variable definitions, two of which look like this:

```
instance_name="@role_name-@region_name"
master_exec_path="/net/dist/bin/merger"
```

The `instance_name` definition assigns the name that MAPmaker will use to refer to a given merger instance in diagnostic logs and status reports. The right-hand side of this assignment refers to two other parameters, `role_name` and `region_name`, which MAPmaker expands when instantiating a merger instance. As a result, each merger instance is given a name with a suffix that identifies the region served. The `master_exec_path` tells MAPmaker the Linux path on the master host where the merger's master executable can be found. The merger class definition also includes a constructor that initializes four variables obtained from data structures external to the merger—for example, it is through the constructor that the name of the merging region is supplied. This example illustrates how the same class definitions can apply to multiple instances of

[1] The only DIST communications *outside* these ssh channels are the traffic forwarded by Saluki (see Section 4.1) and the feedback channel that updates Saluki's sampling policies, for example, channel-sampling and refocusing. Both are encrypted.

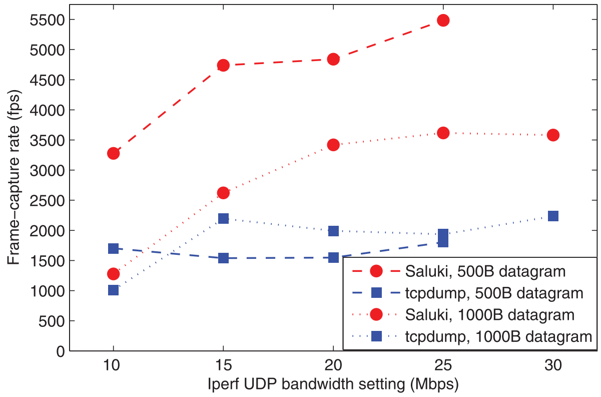


Fig. 5. Comparison of frame-capture rate.

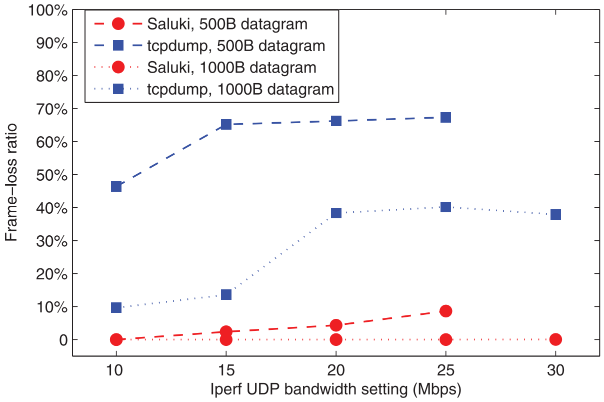


Fig. 6. Comparison of frame-loss ratio.

a program even in the case of parameters whose concrete values vary from one instance to the next.

5 Evaluation

We undertook two classes of experiments to evaluate DIST: first, a controlled-environment evaluation, in which we evaluated Saluki's extreme performance; second, a real-world in-production evaluation, in which we ran the complete DIST system to monitor Dartmouth's production WLAN.

5.1 Controlled Environment Evaluation

In this section, we evaluate Saluki in terms of memory usage, CPU usage, frame-capture rate, and frame-loss ratio. Because tcpdump, Kismet and dingo are all built on libpcap, and tcpdump is the simplest (and should also be the fastest) among them, we used tcpdump as the baseline for comparison. To release tcpdump's maximum potential [37], we directed its output to `/dev/null` instead of the screen or a file.[2] We set the capture size for tcpdump and Saluki to 192 bytes.

We set up two laptops (each a Thinkpad T42 with 1.6-GHz Pentium M CPU and 1.5-GB RAM) to act as the IEEE 802.11g access point and the client, respectively. These two laptops were placed about 2 m (6 feet) from each other, and one Aruba AP70 sniffer was placed halfway between them. We used Iperf [38] as the traffic generator running on two laptops.

We used the Linux command "top" to query memory usage. During execution, Saluki occupied 660-KB RAM, and tcpdump used 740-KB RAM. Note that, since tcpdump is dynamically linked with libpcap, its actual memory usage would be larger than 740 KB if the memory used by libpcap were counted. Of the 660-KB RAM consumed by Saluki, a substantial amount was allocated to various buffers for better performance. For example, the size of the second ring buffer (connecting Threads 1 and 2) was about 90 KB, and the sizes of the compression and encryption buffers were about 30 KB each. If needed, one can reduce Saluki's memory usage by shrinking these buffers.

Figs. 5, 6, and 7 show the performance in terms of frame-capture rate, frame-loss ratio, and CPU usage. The frame-capture rate measures the speed that a sniffing program captures frames in fps. The frame-loss ratio is the ratio of *the number of lost frames reported by the OS kernel* to *the sum of the number of captured frames and lost frames*. Since Saluki is a multithreaded program, its CPU usage in Fig. 7 is the sum of all its threads' usage.

We used Iperf to generate constant-bit-rate UDP traffic with 500B (500-byte) and 1,000B (1,000-byte) datagrams under five UDP bandwidth settings: 10, 15, 20, 25, and 30 Mbps. Three things are worth noting. First, our purpose was to explore each system's response to various traffic loads; 30 Mbps is close to the maximum possible throughput. Second, these five bandwidth numbers are the parameters given to Iperf; in reality, the actual bandwidth could be a bit lower than the setting because of noise and collisions in the (usually quiet) Wi-Fi channel. Third, for a given bandwidth setting, Iperf needed to generate many more small-size packets than large ones to achieve that bandwidth. Due to the limited CPU power on the laptop, we could not generate sufficient 500B UDP packets to reach 30 Mbps. Thus, we do not provide a result for that setting. Each experiment ran for 200 s.

Fig. 5 shows that Saluki captured frames much faster than tcpdump under all settings even though Saluki needed to complete much more work (data compression, data encryption, and UDP forwarding) than tcpdump. Saluki's advantage became more obvious when dealing with high-speed traffic. When Saluki captured 5,488 fps, tcpdump

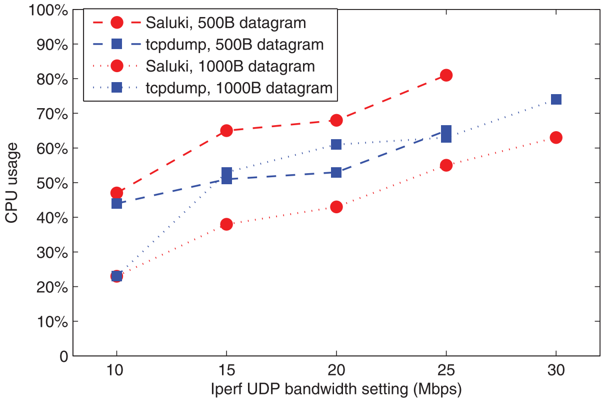


Fig. 7. Comparison of CPU usage.

[2] That is, `tcpdump -i ath0 -n -s 192 -w/dev/null`.

only captured 1,802 fps. In this case, Saluki captured more than three times as many frames as tcpdump did.

Fig. 6 demonstrates that Saluki's frame-loss ratio was significantly lower than tcpdump's. For UDP traffic with 1,000B datagrams, Saluki's frame-loss ratio was nearly always zero (except for 0.028 percent under 30 Mbps), while tcpdump could lose around 40 percent of frames. For UDP traffic with 500B datagrams, the disparity was more obvious (8.6 percent versus 67.4 percent in the worst case).

We make the following interesting observation: by comparing "tcpdump, 1,000B datagram" to "tcpdump, 500B datagram" in Fig. 5, we can see that tcpdump usually captured 500B frames at a lower rate than it captured 1,000B frames, even though Iperf sent them at a higher rate. In Fig. 6, one can see that tcpdump lost a much higher fraction of 500B frames. We speculate that tcpdump dropped many "half-processed" frames when new frames arrived too quickly.

Fig. 7 summarizes Saluki and tcpdump's CPU usage. When there was not much traffic, their CPU usages were comparable. When traffic volume was high, Saluki's CPU usage was higher than tcpdump's. Considering Saluki captured more than three times as many frames and included other work, this amount of increased CPU usage, however, is reasonable.

It is worth noting that the above evaluation results were achieved when the optional HMAC computation was turned off. When this feature was turned on, no noticeable performance changes (frame-capture rate, frame-loss ratio) were observed except that Saluki's CPU usage was increased by about 7 percent (from around 80 to around 87 percent) under the busiest evaluation condition (25-Mbps UDP, 500B datagram).

5.2 In-Production Evaluation

To evaluate the performance of DIST, we continuously monitored Dartmouth's production wireless network for 62 days (from January 4, 2011 to March 6, 2011); 206 out of 210 AMs were used for this evaluation (the remaining 4 AMs were reserved for debugging purposes). We ran Saluki on both radio interfaces of each AM, in total providing 412 Saluki instances on 412 radio interfaces. To cover the 11 IEEE 802.11b/g channels, we configured Saluki to use equal-time channel sampling. Saluki would jump to a new channel after dwelling on one channel for 0.2 s, and thus one iteration took 2.4 s. For this evaluation equal-time sampling was considered sufficient, although coordinated sampling is usually employed for longer term monitoring. To minimize the capturing overlap between two radio interfaces on the same AM, the two Saluki instances running were set to listen to different channels at any given time, leaving a channel distance of six between the radios. While our Aruba AP70s did not support the many additional channels provided by IEEE 802.11n, our software could easily be extended to support these. Table 3 summarizes the experiment configuration.

TABLE 3
In-Production Experiment Configuration

Start time	00:00, Jan 4, 2011
Duration	62 days
Radio interfaces	412
AMs	206
Channel sampling	equal-time sampling
Sampling interval	0.2 seconds
Channels monitored	11 802.11b/g channels
Collected trace file size	3.7 terabytes (gzip compressed)

We ran a single instance of DISTSANI on a server to receive and process all traffic captured from 412 Saluki instances. This server has two 3.0-GHz Intel Xeon CPUs and 4-GB RAM. DISTSANI wrote the processed network traces in pcap file format to a 6-terabyte RAID attached to this server. It is worth noting that, for privacy reasons, Dartmouth only allows us to save IEEE 802.11 frame headers (no IP, TCP/UDP headers) to persistent storage [18]. In total, 3.7 terabytes of compressed pcap trace files were generated in this 62-day experiment (in uncompressed form, these trace files would occupy about 24 terabytes). DISTSANI's CPU usage was between 14 percent and 25 percent during the entire experiment. With the HMAC-SHA256 turned on, an extra 2.40-4.49 percent CPU usage would be added to the above numbers. Such low CPU usage validates DISTANI's online processing capability in a production environment.

Fig. 8 gives an overall picture about the frames processed by the DIST system. The top subplot shows the frame rate of DIST combo frames received by DISTSANI. Since each DIST combo frame may carry hundreds of IEEE 802.11 frames, the bottom subplot of Fig. 8 shows the frame rate of the IEEE 802.11 frames encapsulated in the received DIST combo frames. In Figs. 8 and 9, we use a "boxplot" style to project all measurement results in this 62-day experiment onto a typical 24-hour-calendar-day axis. Each box in a boxplot depicts 5-number summaries for each nonoverlapping 15-minute time window: the upper quartile (the top edge of a blue rectangle), the lower quartile (the bottom edge of a blue rectangle), the median (the red line between the top and bottom edges), the maximum (the top point of the black-dotted whisker line), and the minimum (the bottom point of the black-dotted whisker line). Here we set the maximum whisker length to 1.5, and thus the red crosses lying outside of any box are outliers which imply that they are out of 99.3 percent coverage if the data are normally distributed.

In Fig. 8, the frame rate of DIST combo frames was relatively stable over time: mainly between 410 fps (frames per second) and 440 fps. Since there were 412 Saluki instances, we observed that each Saluki instance transmitted about one DIST combo frame per second, corresponding well to the preset 1-s frame-holding threshold (see Section 4.1.2). Most of the time our DIST system worked in a light-load condition in which most Saluki instances were not busy and forwarded DIST combo frames to servers triggered by the 1-s timer instead of a full combo frame. This result demonstrates our DIST system's efficiency and scalability in a practical network monitoring environment.

The frame rate of IEEE 802.11 frames varied more. The maximum frame rate (42,727 fps) was 1.73 times the minimum (24,699 fps). Moreover, the distance between the upper and lower quartiles, that is, the height of the blue rectangles, varied more than that of DIST combo frames. It reflected the dynamics of the monitored network and

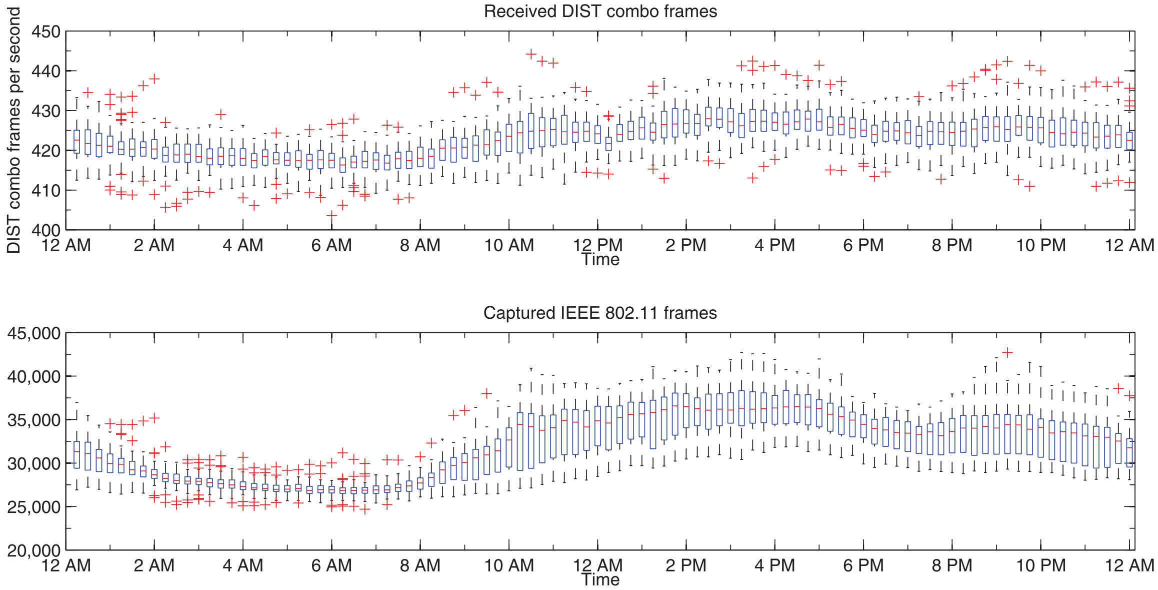


Fig. 8. Captured 802.11 frames and received DIST combo frames.

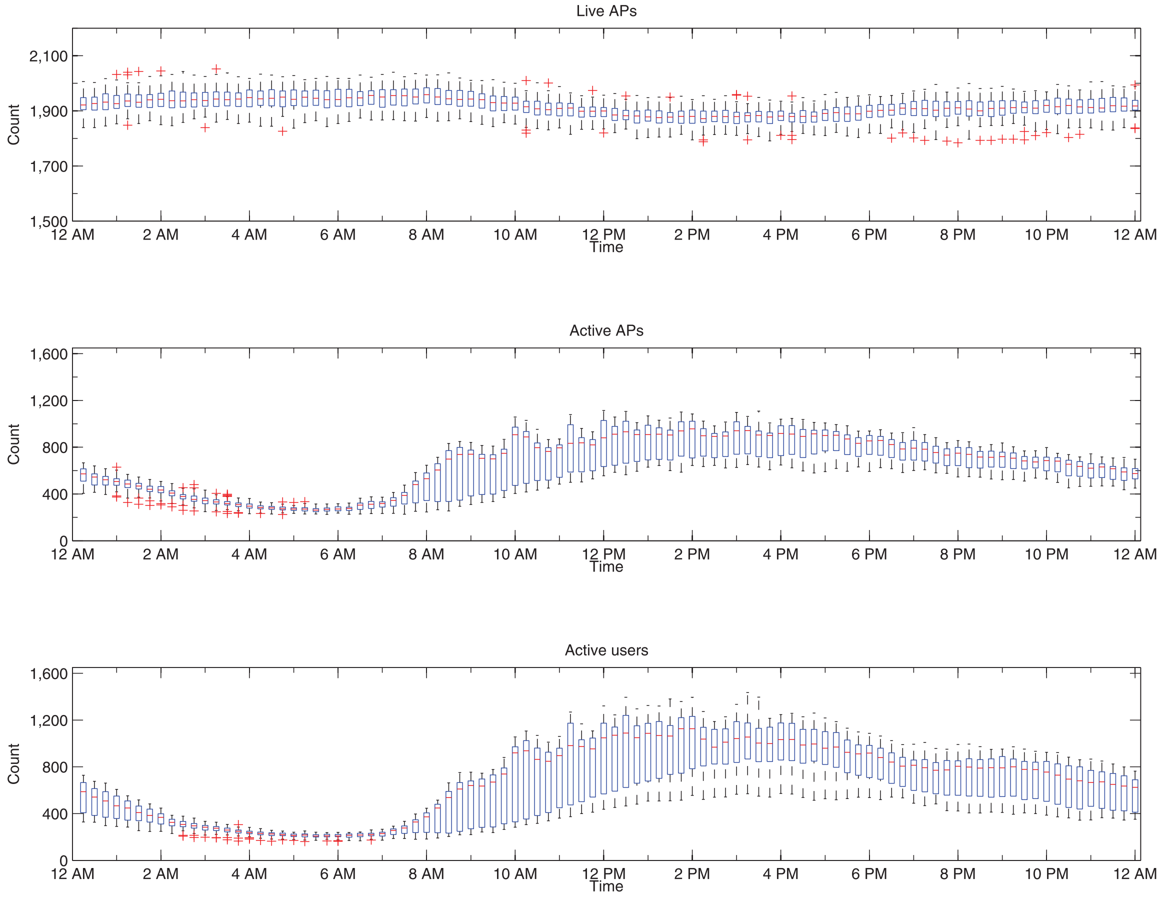


Fig. 9. Live/active APs and active users.

followed a diurnal pattern: fewer frames in the early morning and more frames during daytime.

Fig. 9 examines the number of live and active APs and the number of active users over the same period. Each box summarizes the distribution of average values computed over non-overlapping 15-minute windows. A *live AP* is considered to be one distinct wireless interface (identified by a distinct MAC address) transmitting beacon frames, and an *active AP* is a live AP that is transmitting or receiving data frames. Because one physical AP can generate multiple virtual APs and each virtual AP has a distinct MAC address, the number of live and active APs summarized in Fig. 9 may be bigger than the number of physical APs installed on our campus. An active user is a wireless card that is exchanging data frames with an AP. It is possible that a wireless card may have been used in multiple devices, a device has been used by multiple people, or a person may have multiple wireless devices, but we equate "active card" with "active user" for the simplicity of expression. Two interesting observations can be drawn from Fig. 9. First, the variation of active users shown in Fig. 9 followed a diurnal pattern as seen in Fig. 8. Second, the ratio between the active APs and the live APs was low. Even at the peak time, only about 60 percent of APs were actively used. This result implies that Dartmouth's production wireless network has substantial redundancy for coverage reasons, and in the future it may be possible to employ some energy-saving management strategies without jeopardizing the user experience.

6 Applications Enabled by DIST

As discussed in Section 4.1, Saluki reduces the CPU demands on each AM. Each AM can now collect a more faithful network trace under high traffic loads, or undertake additional tasks providing more fine-grained monitoring or protection of the wireless network. Moreover, as our Aruba AP70 monitors have two radio cards, we can collect traffic with one while transmitting protective or interrogative frame sequences with the other.

In this section, we detail two representative applications of DIST, made possible by the technical improvements of DIST over MAP.

6.1 Distribution-Based Layer 2 Monitoring

Network traffic monitoring is essential to maintain network security, and to troubleshoot problems such as those due to misconfiguration. Hardware manufacturers are beginning to provide some solutions to enable monitoring, for example, Cisco CleanAir [39]. Since monitoring each packet that flows through the network is rather expensive, metrics measuring aggregate statistics of traffic are used to summarize the traffic instead. Choosing good metrics that reveal useful characteristics of the traffic is a hard problem, requiring a tradeoff between the amount of detail gleaned about the traffic and the amount of information that can be considered in a timely manner. Recently, the distribution-based metric of entropy has received increasing research attention [40], [41], [42], [43].

Distributional metrics such as entropy are aggregate functions of the distribution (histogram) of traffic features (usually frame fields) in any given time interval. Lakhina et al. [40] and Nychis et al. [41] showed the usefulness of monitoring the entropy of network traffic for detecting anomalies such as those due to network intrusion, malfunction, or user behavior. We show here that monitoring the relations between pairs of features, via the metric of conditional entropy, is also useful for anomaly detection in wireless traffic streams.

We briefly present a summary of our experiment results of distribution-based approaches to wireless traffic monitoring using the DIST infrastructure. For more detail, see dissertation [44]. The basic approach is similar to that applied in previous work [40], [41], [42]. We partition the observed traffic into successive intervals of equal time length. Within each interval, we extract values of the feature in which we are interested (e.g., BSSID) to get a stream (a sequence) of numbers (BSSIDs). We next consider the distribution induced by frequencies of values in this stream; we compute metrics that summarize each time interval of the stream, and the values of a metric from successive intervals produce a time series. We then employ threshold-based change-point detection methods on this time series to detect anomalies. While we consider several metrics in our work that capture different aspects of the "shape" of the distribution, here we present results only using the number of distinct elements in the distribution. The entropy H of a distribution on n elements with element i $(1 \leq i \leq n)$ having frequency f_i is given by $H = -\sum_{1\leq i\leq n}(f_i/m)\log_2(f_i/m)$, where $m = \sum_i f_i$ is the length of the stream. The number of distinct elements, denoted as F_0, is the number of elements i having nonzero frequency f_i. It is given by $F_0 = \sum_{1\leq i\leq n} f_i^0$, assuming $0^0 = 0$. Both metrics capture different aspects of the "shape" of the distribution.

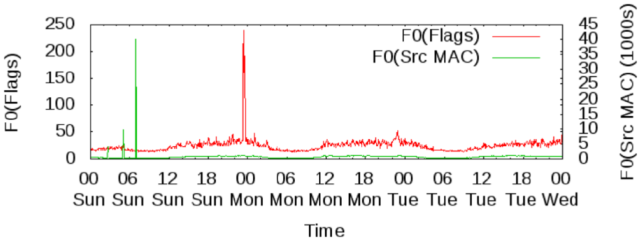


Fig. 10. Timeseries of number of distinct flags, and distinct source MACs.

To evaluate this method, we collected a 3-day data set of wireless traffic between May 2 and May 4, 2010. For this experiment, we consider data from 52 AMs located in the Dartmouth College Library. During this period, we injected four types of anomalous traffic using standard tools from the BackTrack Linux distribution [45] and the Metasploit framework [46]. Our tool, *Baffle* [47], transmits frames with varying values in the flags field, to fingerprint the chipset, firmware, and driver combination of a device suspected of being a rogue AP; *Beacon frame fuzzer* injects beacon frames with malformed field values in an attack against wireless drivers and devices; *probe response frame fuzzer* injects malformed fields in probe response frames; *FakeAP* advertises fake access points using random SSIDs and BSSIDs.

We captured wireless traffic, including the anomalous traffic, and identified the expected anomalies using several features: BSSID, source and destination MACs, frame length, sequence number, flags, and frame type.

We can clearly identify the anomalies in the plots in Fig. 10. As expected, Baffle results in several new flag combinations being observed (near 00/Mon), and the fuzzing and fake AP anomalies result in many new source MAC addresses (near 06/Sun).

We are continuing to investigate several aspects of this work, including the choice of specific features and metrics for monitoring, parameter tuning, criteria for drawing inferences, and investigating the computational challenges involved. Monitoring distributional metrics in a computationally efficient manner is a crucial, nontrivial task given the high rates of traffic produced by DIST. The problem is exacerbated further due to the fact that monitoring of multiple traffic features is usually desired. The algorithms that naturally come to mind are infeasible due to constraints imposed on both memory and computation time for online monitoring. Earlier, we developed some strategies to deal with this computational challenge [42]. The choice of our metrics is influenced by the availability of recently developed efficient online algorithms to compute them [48], [49].

6.2 Active Protection System (APS)

As discussed above, Saluki reduces the CPU demands on each AM. Each AM can now collect a more faithful network trace under high traffic loads, or undertake additional tasks providing more fine-grained monitoring or protection of the wireless network. Moreover, as our Aruba AP70 monitors have two radio cards, we can collect traffic with one while transmitting protective or interrogative frame sequences with the other.

In this section, we detail one representative application: an APS; details in [26]. While many techniques have been developed to detect potential security threats on a wireless network, there are few techniques to *protect* normal users against these threats. For a wired network, a network administrator can block a station's access to the network by port blocking, but this is not possible for a wireless network due to the open medium. An adversary can send whatever he wants over the air; even if a wireless IDS can detect the malicious behavior, it cannot stop it. The goal of the DIST APS is to provide the wireless network administrator with a tool to mitigate the ongoing security threats.

6.2.1 Implementation

In its first version, our APS focuses on the unauthorized AP threat. Due to security concerns, an enterprise WLAN administrator often requires the users to connect only to an enterprise-controlled AP. However, an unauthorized AP (either a rogue AP or an impersonation of an AP [50]) can easily breach this security policy. Our APS uses several denial-of-service (DoS) attacks in a "benign" way that prevents users from connecting to an unauthorized AP, or forces them to break an existing connection. The APS has two components: a back-end controller and a front-end agent. The APS controller running on a backbone server monitors the alerts generated by DIST detectors, each of which is monitoring the stream of frames captured by AMs, looking for evidence of unauthorized APs. After an alert is received, the APS controller takes two steps: 1) it compares the target MAC address to the whitelist of known-but-not-ours APs. If the target MAC address is in the whitelist, the APS assumes this is a legitimate AP and does not take subsequent steps. Otherwise, APS will proceed to the next step: 2) it determines the actions (according to predefined rules) and the set of AMs that should participate. Then it composes and sends commands to the APS front-end agent running on each involved AM. An APS command includes at least the following four fields: target MAC address, channel, action, and duration (such a command is protected by the Rabbit cipher and HMAC-SHA256 to ensure confidentiality, authenticity, and integrity). The APS agent parses the command, prepares the interface and launches the attack against the specified target. It is worth noting that, even though an adversary can move to other channels or other places when he discovers the APS, it is difficult to bypass the system entirely because of the full-spectrum coverage and broad deployment of DIST. Currently, the APS agents are running on DIST AMs, but technically it is possible to integrate them into production APs to save cost.

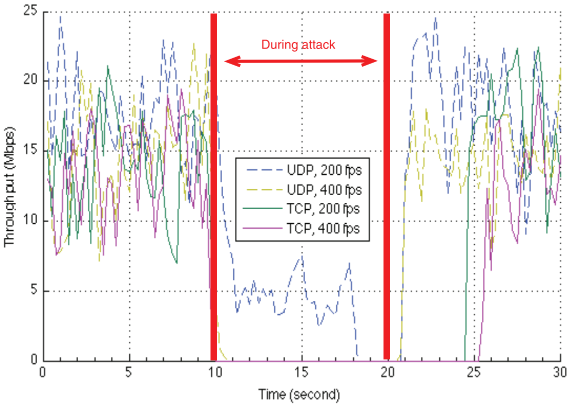


Fig. 11. Evaluation of deauthentication DoS attack.

6.2.2 Evaluation

We evaluated the DIST APS using the metrics of response time and protection effectiveness.

Response time. A quick reaction is important to protect users from unauthorized APs. The response time is the time between the moment when the APS controller receives an alert and the moment when the APS agent transmits its first attack frame. Because our AMs are deployed around Dartmouth campus, the response time is the sum of 1) the APS controller's processing time, 2) the network delay, and 3) the APS agent's processing time. The measured average network delay was 0.372 ms, and the average processing time for APS controller and agent were 0.025 and 350.347 ms, respectively. The total response time was thus about 351 ms.

Protection effectiveness. We employed two DoS attacks to interrupt the operation of an unauthorized AP: a Queensland DoS attack [51] and a deauthentication attack [52]. A Queensland DoS attack is a blind-jamming attack that can disable all Wi-Fi activity on the given channel in the immediate vicinity. In our lab environment, we observed that its effective radius reached at least 50 feet. Compared to the Queensland DoS attack, the deauthentication attack is more "intelligent" because it will only be effective on the target device and will not interfere with other devices on the same channel.

Fig. 11 shows how the deauthentication attack affects both UDP and TCP traffic under different attack intensities. The purpose of this experiment is not only to evaluate the effectiveness with which the deauthentication attack disconnects users from unauthorized APs, but more importantly it is to estimate the cost for APS to successfully launch such an attack. To simulate a busy channel, we tried to transmit as much UDP and TCP traffic as possible. Obviously, UDP traffic was much more robust than TCP traffic against DoS attack. The attack was launched between $t = 10$ and $t = 20$ in this plot. In Fig. 11, a 200-fps deauthentication attack completely blocked the TCP traffic while, for UDP, it required 400 fps to do so. We also observed that UDP recovered more quickly than TCP from the attack: 1 s versus 4-5 s. From the cost perspective, because one deauthentication frame is only 58 bytes, one APS agent only occupies 185.6 KBps (≈ 1.5 Mbps) bandwidth when sending 400 deauthentication frames per second. Such an attack can protect all nearby clients from the unauthorized AP, no matter how many clients.

7 Summary

As an important part of the Internet edge, enterprise-wide WLANs are increasingly used for many mission-critical tasks. Monitoring such WLANs is important to understanding the performance and security aspects of the Internet experience. However, monitoring a large-scale WLAN is a difficult undertaking. In this paper, we introduce the design, implementation and evaluation of DIST, a large-scale general-purpose WLAN monitoring system. As the successor of MAP, DIST has faced many challenges related to efficiency, scalability, security, and management. Saluki, DISTSANI, and MAPmaker are our solutions to these challenges. The combined strength of these subsystems makes DIST an efficient and scalable WLAN monitoring system, which has been validated by both controlled and real-world evaluations. Although Saluki, DISTSANI, and MAPmaker have been designed to fit the special needs of DIST, they are also applicable to general WLAN measurement tasks with variable scales.

DIST provides us a unique platform to study a large-scale WLAN and its users. DIST's wide coverage facilitates community-oriented network research, such as how a large body of users uses the network and how the users interact with each other. We also built a system that uses the high-resolution data captured by DIST to help the computing service at Dartmouth to diagnose malfunctions, and detect any abnormal behaviors. Furthermore, we studied the obstacles and tradeoffs in sanitizing network traces [7].

Acknowledgments

This paper results from a research program in the Institute for Security, Technology, and Society (ISTS), supported by the US Department of Homeland Security under Grant Award Number 2006-CS-001-000001 and by the NetSANI project at Dartmouth College, funded by Award CNS-0831409 from the US National Science Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the US Department of Homeland Security or the US National Science Foundation.

References

[1] D. Kotz and K. Essien, "Analysis of a Campus-Wide Wireless Network," *Wireless Networks,* vol. 11, nos. 1/2, pp. 115-133, Jan. 2005.

[2] T. Henderson, D. Kotz, and I. Abyzov, "The Changing Usage of a Mature Campus-Wide Wireless Network," *Computer Networks,* vol. 52, no. 14, pp. 2690-2712, Oct. 2008.

[3] U. Deshpande, C. McDonald, and D. Kotz, "Refocusing in 802.11 Wireless Measurement," *Proc. Passive and Active Measurement Conf. (PAM '08),* Apr. 2008.

[4] U. Deshpande, C. McDonald, and D. Kotz, "Coordinated Sampling to Improve the Efficiency of Wireless Network Monitoring," *Proc. IEEE 15th Int'l Conf. Networks (ICON),* Nov. 2007.

[5] Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell, "Detecting 802.11 MAC Layer Spoofing Using Received Signal Strength," *Proc. IEEE INFOCOM,* Apr. 2008.

[6] Y. Sheng, G. Chen, H. Yin, K. Tan, U. Deshpande, B. Vance, D. Kotz, A. Campbell, C. McDonald, T. Henderson, and J. Wright, "MAP: A Scalable Monitoring System for Dependable 802.11 Wireless Networks," *IEEE Wireless Comm.,* vol. 15, no. 5, pp. 10-18, Oct. 2008.

[7] K. Tan, G. Yan, J. Yeo, and D. Kotz, "Privacy Analysis of User Association Logs in a Large-Scale Wireless LAN," *Proc. IEEE INFOCOM,* Apr. 2011.

[8] A. Balachandran, G.M. Voelker, P. Bahl, and P.V. Rangan, "Characterizing User Behavior and Network Performance in a Public Wireless LAN," *SIGMETRICS Performance Evaluation Rev.,* vol. 30, no. 1, pp. 195-205, 2002.

[9] M. Afanasyev, T. Chen, G.M. Voelker, and A.C. Snoeren, "Analysis of a Mixed-Use Urban Wi-Fi Network: When Metropolitan Becomes Neapolitan," *Proc. ACM SIGCOMM Conf. Internet Measurement (IMC),* 2008.

[10] Y.-C. Cheng, J. Bellardo, P. Benkö, A.C. Snoeren, G.M. Voelker, and S. Savage, "Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis," *SIGCOMM Computer Communication Rev.,* vol. 36, no. 4, pp. 39-50, 2006.

[11] P. Bahl, R. Chandra, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill, "Enhancing the Security of Corporate Wi-Fi Networks Using DAIR," *Proc. ACM MobiSys,* 2006.

[12] V. Kone, M. Zheleva, M. Wittie, B.Y. Zhao, E.M. Belding, H. Zheng, and K. Almeroth, "AirLab: Consistency, Fidelity and Privacy in Wireless Measurements," *SIGCOMM Computer Comm. Rev.,* vol. 41, pp. 60-65, Jan. 2011.

[13] "Aruba Networks," http://www.arubanetworks.com, 2013.

[14] "OpenWrt," http://openwrt.org, 2013.

[15] U. Deshpande, T. Henderson, and D. Kotz, "Channel Sampling Strategies for Monitoring Wireless Networks," *Proc. Second Int'l Workshop Wireless Network Measurement (WiNMee),* Apr. 2006.

[16] M. Raya, J.-P. Hubaux, and I. Aad, "DOMINO: Detecting MAC Layer Greedy Behavior in IEEE 802.11 Hotspots," *IEEE Trans. Mobile Computing,* vol. 5, no. 12, pp. 1691-1705, Dec. 2006.

[17] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Analyzing the MAC-Level Behavior of Wireless Networks in the Wild," *SIGCOMM Computer Comm. Rev.,* vol. 36, no. 4, pp. 75-86, 2006.

[18] S. Bratus, D. Kotz, K. Tan, W. Taylor, A. Shubina, B. Vance, and M.E. Locasto, "Dartmouth Internet Security Testbed (DIST): Building a Campus-Wide Wireless Testbed," *Proc. Workshop Cyber Security Experimentation and Test (CSET),* Aug. 2009.

[19] "Nagios - The Industry Standard in IT Infrastructure Monitoring," http://www.nagios.org, 2013.

[20] "Cacti: The Complete RRDTool-Based Graphing Solution," http://www.cacti.org, 2013.

[21] "TCPDUMP/LIBPCAP Public Repository," http://www.tcpdump.org, 2013.

[22] "Wireshark," http://www.wireshark.org, 2013.

[23] "Kismet," http://www.kismetwireless.net, 2013.

[24] "MadWifi Project," http://madwifi-project.org, 2013.

[25] K. Tan and D. Kotz, "Saluki: A High-Performance Wi-Fi Sniffing Program," *Proc. Int'l Workshop Wireless Network Measurements (WiNMee),* May 2010.

[26] K. Tan, "Large-Scale Wireless Local-Area Network Measurement and Privacy Analysis," PhD dissertation, Dartmouth College, http://www.cs.dartmouth.edu/reports/TR2011-703.pdf, Aug. 2011.

[27] "Linux Packet MMap," http://wiki.ipxwarzone.com/index.php5?title=Linux_packet_mmap, 2013.

[28] T.A. Welch, "A Technique for High-Performance Data Compression," *Computer,* vol. 17, no. 6, pp. 8-19, 1984.

[29] "QuickLZ," http://www.quicklz.com, 2013.

[30] "FastLZ," http://www.fastlz.org, 2013.

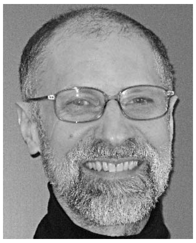
[31] M. Robshaw, "The eSTREAM Project," *New Stream Cipher Designs: The eSTREAM Finalists,* 2008.

[32] P. Ekdahl and T. Johansson, "A New Version of the Stream Cipher SNOW," *Proc. Revised Papers from the Ann. Int'l Workshop Selected Areas in Cryptography (SAC),* 2003.

[33] "The Keyed-Hash Message Authentication Code (HMAC)," *Information Technology Laboratory at NIST,* http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf, 2011.
[34] O. Gay, "HMAC-SHA2," http://www.ouah.org/ogay/hmac/, 2013.
[35] K. Tan, J. Yeo, M.E. Locasto, and D. Kotz, "Catch, Clean, and Release: A Survey of Obstacles and Opportunities for Network Trace Sanitization," *Privacy-Aware Knowledge Discovery: Novel Applications and New Techniques,* F. Bonchi and E. Ferrari, eds., Chapman and Hall/CRC, Dec. 2010.
[36] A.G. Miklas, S. Saroiu, A. Wolman, and A.D. Brown, "Bunker: A Privacy-Oriented Platform for Network Tracing," *Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI),* Apr. 2009.
[37] "Libpcap with MMAP," http://public.lanl.gov/cpw, 2013.
[38] "Iperf," http://sourceforge.net/projects/iperf, 2013.
[39] "Cisco Cleanair," http://www.cisco.com/go/cleanair, 2013.
[40] A. Lakhina, M. Crovella, and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," *SIGCOMM Computer Comm. Rev.,* vol. 35, no. 4, pp. 217-228, 2005.
[41] G. Nychis, V. Sekar, D.G. Andersen, H. Kim, and H. Zhang, "An Empirical Evaluation of Entropy-Based Traffic Anomaly Detection," *Proc. ACM SIGCOMM Conf. Internet Measurement (IMC),* 2008.
[42] C. Arackaparambil, S. Bratus, J. Brody, and A. Shubina, "Distributed Monitoring of Conditional Entropy for Anomaly Detection in Streams," *Proc. IEEE Workshop Scalable Stream Processing Systems (SSPS),* 2010.
[43] W. Lee and D. Xiang, "Information-Theoretic Measures for Anomaly Detection," *Proc. IEEE Symp. Security and Privacy (S&P),* 2001.
[44] C. Arackaparambil, "Anomaly Detection in Network Streams through a Distributional Lens," PhD dissertation, Dartmouth College, http://www.cs.dartmouth.edu/reports/TR2011-707.pdf, Sept. 2011.
[45] "Backtrack Linux," http://www.backtrack-linux.org, 2013.
[46] "The Metasploit Project," http://www.metasploit.com, 2013.
[47] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles, "Active Behavioral Fingerprinting of Wireless Devices," *Proc. ACM Conf. Wireless Network Security (WiSec),* 2008.
[48] N. Alon, Y. Matias, and M. Szegedy, "The Space Complexity of Approximating the Frequency Moments," *Proc. Ann. ACM Symp. Theory of Computing (STOC),* 1996.
[49] N.J.A. Harvey, J. Nelson, and K. Onak, "Sketching and Streaming Entropy via Approximation Theory," *Proc. IEEE Ann. Symp. Foundations of Computer Science (FOCS),* 2008.
[50] R. Beyah and A. Venkataraman, "Rogue-Access-Point Detection: Challenges, Solutions, and Future Directions," *IEEE Security Privacy,* vol. 9, no. 5, pp. 56-61, Sept./Oct. 2011.
[51] AUSCERT Advisory, "Denial of Service Vulnerability in IEEE 802.11 Wireless Devices," http://www.auscert.org.au/render.html?it=4091, 2013.
[52] J. Bellardo and S. Savage, "802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," *Proc. USENIX Security Symp.,* Aug. 2003.
[53] "Community Resource for Archiving Wireless Data at Dartmouth (CRAWDAD)," http://www.crawdad.org, 2013.

Keren Tan received the PhD degree in computer science from Dartmouth College in 2011. His research interests include machine learning and pattern recognition, network system measurement and optimization, and cloud computing. Currently, he is working at F5 Networks, Inc.

Chris McDonald received the BSc degree in computer science and mathematics and the PhD degree in computer science both from the University of Western Australia. He currently holds the appointments of associate professor in the School of Computer Science and Software Engineering at the University of Western Australia (UWA) and adjunct associate professor in the Department of Computer Science at Dartmouth College, New Hampshire. He has recently taught in the areas of computer networking, security and privacy, mobile and wireless computing, software design and implementation, C programming, and operating systems at UWA and Dartmouth. Together with these areas, his research interests include wireless, ad hoc, and mobile networking; network simulation; and computer science education. He is a member of the ACM.



Bennet Vance received the bachelor's degree in mathematics from Yale in 1976 and graduate degrees in computer science from Stanford and from the OGI School of Science and Engineering in 1981 and 1998, respectively. He has worked as a software developer and consultant and has held positions at AT&T Bell Laboratories, the IBM Almaden Research Center, and in several departments at Dartmouth College.

Chrisil Arackaparambil received the PhD degree in computer science from Dartmouth College in 2011. He is currently a software engineer working on the ZFS file system at Oracle, Inc.

Sergey Bratus is a research assistant professor in the Dartmouth College Computer Science Department. His primary research interests include practical defensive and offensive security.

David Kotz received the AB degree in computer science and physics from Dartmouth in 1986, and the PhD degree in computer science from Duke University in 1991. After that, he returned to Dartmouth to join the faculty. He is the Champion International Professor in the Department of Computer Science and an associate dean of the Faculty for the Sciences at Dartmouth College. During the 2008-2009 academic year, he was a Fulbright research scholar at the Indian Institute of Science, Bangalore, India. At Dartmouth, he was the executive director of the Institute for Security Technology Studies from 2004-2007. His research interests include security and privacy, pervasive computing for healthcare, and wireless networks. He has published more than 100 refereed journal and conference papers. He is a fellow of the IEEE, a senior member of the ACM, a member of the USENIX Association, and an elected member of Phi Beta Kappa.