Constructing a High-Rate Covert Communication Channel in RDMA-Equipped Systems

Plamen Petrov

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

In present days, clouds store vast amounts of data. Government agencies and large companies have their private clouds to carry out confidential research and to store sensitive data. Recently, a number of covert channels have been discovered that enable the covert communication in the cloud between parties such as two virtual machines on the same host machine. These attacks violate the isolation guarantees which are assumed in the cloud. As a result, covert channels pose a significant threat to the containment and protection of confidential data stored in clouds.

In this work, we present the Bankrupt attack. It is a covert channel which enables the communication between a sender and a receiver in an RDMA-connected cluster. We assume that the sender and the receiver are not allowed to interact with each other over a legitimate communication channel and are located on different physical machines. To establish the covert communication, we create a timing channel in one of the memory banks of a third machine in the cluster that both the sender and the receiver can access via RDMA.

The Bankrupt attack poses a security threat as it can be used to exfiltrate information from a machine in a private cloud which is not connected to the Internet to another machine in the same cluster, which has access to the public Internet. From there, the secret can be transferred into the hands of a malicious entity.

In the report, we elaborate on the design of the covert channel. We provide a procedure to identify a set of virtual memory addresses which map to the same bank in memory. In addition, we give an algorithm to decode the obtained signal. We list a number of mitigation techniques, which aim to enhance existing technology to detect or protect against the Bankrupt channel.

We evaluate the covert channel on a private cluster at the University of Edinburgh and the CloudLab research cluster. We find that on our cluster the channel achieves a throughput of 158.73Kbps and an error rate of 5%. The channel achieves a throughput of 204.5Kbps and an error rate of 3.5% on CloudLab. The channel's throughput and error rate are not affected by local memory load. The covert channel remains operational in the presence of network load, but has a 30% lower throughput and a higher error rate. The channel remains undetected by existing profiling tools, which strengthens its practicability.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Boris Grot for giving me the opportunity to work on this project and supporting me continuously from the start of it till the very end. I want to thank Prof. Grot for not only giving me feedback, but also helping me learn from it.

I would like to thank Dmitrii Ustiugov for his utmost support throughout the year. I want to thank Dmitrii for introducing me to whole idea for the covert channel and for dedicating a significant portion of his time to helping me make progress with the project. Among many other things, I am thankful to Dmitrii for pointing me to relevant resources about memory organization, and for providing valuable insight about the results and how to improve them.

I would like to thank Siavash Katebzadeh for helping me get started with RDMA and for answering all of my questions regarding it. I also want to thank Siavash for providing the scripts to create network load in the cluster.

I would like to thank Artemiy Margaritov for pointing me to the mfc benchmark to evaluate the channel's behaviour under local memory load. I also want to thank Artemiy for pointing me to several memcached clients that I experimented with and for helping me understand the impact of different kernel options on the covert channel.

Last but not least, I would like to thank Antonis Katsarakis and Vasilis Gavrielatos for testing the code on CloudLab and providing feedback.

Table of Contents

Chapter 1

Introduction

1.1 Overview

Nowadays, the increased need for computing has caused most major companies to create their own distributed computing infrastructures. These are often called private clouds or private data centers and they consist of thousands of machines. Private clouds are used to operate what have become important services such as email, mapping services, and search engines. Due to the tight performance and quality of service (QoS) requirements of these applications, the servers in the private cloud often use cutting edge hardware and they are connected using high-end networking. Moreover, large companies deal with enormous amounts of sensitive company and customer data, like credit card numbers, search history, personal communications, and more. Hence, it is of utmost importance that their computing infrastructure is highly protected from intrusions.

Governments also have their private data centers. For example, the Pentagon, the headquarters of the United States Department of Defense (USDoD), uses one to carry out military and other research, to run expensive simulations, to store information about arms, confidential documents, and much more data that is unknown to the general public. Leaking a few such facts would likely cause an international uproar and pose a severe threat to national security not only of the US, but of other countries as well. It would also lead to lengthy investigations into the source and culprits of the breach. For these reasons, the Pentagon and similar institutions in other countries are some of the most heavily guarded facilities in the world, both from a physical and a computer security point of view.

Companies and institutions which posses private data centers employ strict policies to ensure the confidentiality and integrity of their data. Some general policies include strong authentication and encryption. Virtual machines are used to ensure process isolation of different services within single machines. The machines which run secret services in private clouds are normally isolated from the public Internet. This is a further policy to ensure that attackers who have acquired some sensitive information cannot leak it online. Intrusion detection systems are also put in place to sense unusual activities which might indicate a data breach.

Recently, the Pentagon has signed a $10bn contract with Microsoft to move all of its data to the Microsoft Cloud [19]. This is an unprecedented case, where some of the most secretive data on the globe will move to the computing infrastructure of a commercial company. Although Microsoft will reap huge profits from these negotiations, it has also taken on a great deal of responsibility. Not only does Microsoft need to protect the data from external illegitimate accesses, but it also needs to guard it against its own employees.

Covert channels are attacks which enable the exchange of information between a sender and a receiver in a setting where the communication between the two parties is prohibited. A covert channel can be used to circumvent crucial isolation guarantees in the cloud. For example, it has been demonstrated that two virtual machines on the same physical node can communicate with each other, although it is assumed that virtual machines guarantee isolation. Because covert channels do not use an explicit communication channel, they can cause a continuous leak of data, which is hard to detect and remove. As a result, they pose a severe security threat to applications in the cloud.

In this work, we present the Bankrupt attack. It is a covert channel which exploits the organization of main memory and the design of the network in private clouds to enable the communication between machines on the same cluster. While most other covert channel attacks rely on the attacker's capability to collocate their malicious software on the same CPU, the Bankrupt attack we describe allows to extend the collocation requirement to an entire cluster (or even a datacenter-wide) deployment. In order to do this, the Bankrupt attack uses a third non-malicious machine in the cluster that both the sender and the receiver can communicate with.

Looking back at the Pentagon example, if an attacker acquires access to military plans on a machine that does not have Internet access, they can use the Bankrupt channel to transfer the plans to a machine in the same cluster that has access to the Internet. From there, the information can be sent over the Internet to be leaked to the general public or a foreign government. Using a covert channel is crucial in this case, because the machines in the cluster storing the military plans are disconnected from public Internet for security reasons.

Good bandwidth is essential for a covert channel, but covertness is at least as important. As a concrete example, there is usually no real-time requirement to exfiltrate sensitive information from the Pentagon. On the other hand, it is much more desirable for an attacker to remain undetected so that they can leak data for an extended period of time.

We are able to mount the covert channel on our own cluster and a real cloud. Both experimental platforms use recent hardware, which demonstrates Bankrupt's potential impact on modern cloud infrastructures. The Bankrupt attack achieves a throughput of 204.5Kbps and and error rate of 3.5% on real cloud computing infrastructure. It maintains its performance under load in the system and the cluster, achieves high throughput in the presence of network load as well. In addition, it remains undetectable by commonly monitored profiling metrics.

1.2 Contributions

In this work we make the following contributions:

- We discuss how the combination of some design principles in the organization of main memory and RDMA networks makes it possible to create a covert channel in a private cloud.
- We design and implement the covert channel, which we call Bankrupt.
- We implement a decoder for the signal obtained by the receiver in the covert channel.
- We discuss possible mitigations for the attack.
- We evaluate the channel's throughput and error rate in isolation and under load.
- We evaluate the channel's covertness and its ability to remain hidden from monitoring tools.

1.3 Outline

This section outlines the structure of the report.

Chapter 1 is the introduction, where we explain where and how the channel can be used. We also summarize the contributions and the contents of the report.

Chapter 2 presents the background required for the project and the related work. We go over the basics of covert channels, memory organization and RDMA networks. We also place our work into context with respect to other existing covert channels.

Chapter 3 is the methodology, where we describe the assumptions we make for the attack and the tools available to the attacker. We also present the specifications of our experimental platforms.

In **Chapter 4**, we describe the design decisions in the hardware which make the attack possible. We also give an overview of the attack and the associated challenges and how we solve them. We describe the decoding procedure and list potential mitigations.

In **Chapter 5**, we evaluate the properties of the channel under different cluster settings. We also examine the covertness of the attack.

In **Chapter 6**, we give pointers to future work and summarize the results.

1.4 Previous Work Carried Out

I started with the project this year. I changed from my MInf Part 1 project to this project.

Chapter 2

Background and Related Work

The covert channel that we present combines knowledge of covert channels, memory organization and Remote Direct Memory Access (RDMA). This chapter introduces the fundamentals of each one of these concepts required by the attack. The chapter begins with an introduction to covert channels and then describes the memory organization of modern computers, Remote Direct Memory Access (RDMA) as well as the implications that these have for modern systems. We also briefly introduce the notion of tail latency and tail attacks, which is one of the use cases for the Bankrupt attack.

Finally, we place our work in context by comparing it to other existing covert channels, both in local machines and over the network.

2.1 Covert Channels

Covert channels are attacks which enable the exchange of information between two parties where the communication between them is not normally allowed. Some reasons for this prohibition can be process isolation on a local machine, a firewall in a network, or the isolation between virtual machines running on the same host in a cloud. For example, an application may be used for storing confidential healthcare data. In this setting, it has access to sensitive data but (most likely) no access to the Internet. In this case, the application may attempt to find another application (for example, running on the same physical machine) that it can leak the data to by using one of the known covert channels. Hence, covert channels pose a significant security threat, as containing confidential information becomes a challenging problem.

In a covert channel, the information often flows from the sender to the receiver over a channel which is not designed for communication. Covert channels usually exploit a the properties of a shared resource in a system or an infrastructure. Routers, caches, the memory bus, core temperature, and network adapters have all been used to launch covert channels on modern systems.

One important type of covert channel is a timing covert channel. In this scenario, the sender modulates the response time of the shared resource (e.g., memory access time) while the receiver probes and records the response time that it sees. High access

times observed by the receiver can be taken to mean that the sender is sending a 1 and low access times can be taken to mean 0 in a binary channel. Timing channels have been the primary attack and research target due to the rife timing differences that modern hardware components expose. Nonetheless, there are also thermal, sound covert channels, etc.

The typical requirements for a robust covert channel include sufficient bandwidth to transmit the secret in a reasonable time, low error rate, and being undetectable by software and hardware monitoring capabilities. A desirable property for a covert channel is resistance to the load caused by other applications. For example, this can be the load that other applications have on the response time of the shared resource in a timing covert channel.

Constructing a channel with all these characteristics in a wide area deployment can pose a severe security threat. Covert channels exploit timing differences in the hardware are fundamentally tied to performance, e.g. hit/miss timing difference in caches. Protecting against them might cause a noticeable degradation in performance. This makes the job of a computer engineer more difficult, as it requires not only to identify potential security vulnerabilities, but to fix them in a way which does not devalue performance.

2.2 Memory Organization

Memory hierarchy and main memory subsystem. In modern computers, main memory is much slower than the processor. For this reason, virtually all contemporary machines implement a memory hierarchy. Normally, caches sit on top of the memory hierarchy, which enable fast access to hot items. Their details are not relevant to the attack. The caches are followed by main memory, which has rich structure in modern computers.

The main memory in modern systems typically has 2-6 channels, which can be accessed in parallel. The organization of a generic memory channel is shown in Figure 2.1[1]. In turn, each channel has multiple Dual Inline Memory Modules (DIMMs), which normally have 2 ranks, corresponding to each side of the DIMM. A rank consists of banks, which can be accessed in parallel and store the memory contents in a matrix-like structure with rows and columns. To sum up, the main memory structure resembles a distributed system where the banks are logically independent servers. The ability to serve requests from all banks in parallel means that the memory bandwidth can scale linearly with the number of banks. Under normal conditions, the memory traffic is spread relatively evenly across all banks, in a way that the banks are often underutilized [11].

Row buffer. Each bank has a row buffer which can be thought of as a cache that can only hold one bank row. A row which is active in the row buffer is called ”open”. An access to an open row incurs a latency called Column Address Strobe (CAS). Period-

[1]The figure also shows the processor and the memory controller. Their role is described later in the section.

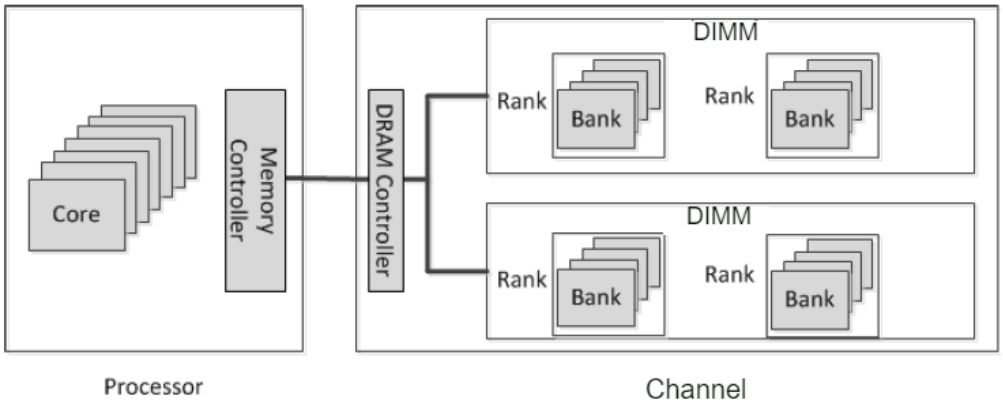


Figure 2.1: Organization of a single memory channel. The processor and the memory controller is also shown. [34]

ically, the bank performs a refresh operation (PRE), where the open row is closed. In that case, if a request comes in to any row when when no row is currently open, the accessed row needs to be loaded with latency known as Row Address Strobe (RAS). In total, the cost of such accesses is RAS + CAS. In the worst scenario, a row which is not open is accessed *while there is a currently open row*. In this case, the open row needs to be closed and the accessed row is fetched to the row buffer. This event is called a row conflict and results in the highest latency that can be incurred in a bank – equal to PRE + RAS + CAS.

Mapping from address to bank. The existence of banks implies that the physical memory of the system needs to be distributed in some way among the banks. Modern processors have one or more integrated memory controllers (IMC) on the chip which coordinate the communication between the memory and the other parts of the system. The memory controller is connected to the memory via a bus. The memory controller and the bus are shown in Figure 2.1. The memory controller is responsible for the mapping from physical addresses to banks. Thus, when a request arrives at the memory controller, it is sent to the respective bank queue based on the mapping function.

The possibility to serve requests from different banks in parallel opens up an opportunity for the memory controller to increase the total memory bandwidth. This can be done by organizing the physical address space in banks in a way that parallel accesses to different banks are maximized. Memory controllers employ a hashing function of some combination of the lower bits of the physical address to map the addresses to a bank. The lower bits are preferred over the higher ones due to their higher entropy, but which exact bits are used depends on the processor as well as the memory configuration – the number of channels, DIMMs, ranks and banks.

Some vendors reveal this mapping, while others like Intel keep it proprietary. Nevertheless, there are tools like Drama [25], which have reverse engineered the hashing function on several Intel processors. Drama forms sets of addresses which belong to the same bank but different rows by exploiting the timing difference that arises from

row conflicts. The mapping functions obtained based on these sets for different configurations can be found in [25].

Even though overall memory bandwidth scales with the number of banks, the throughput of a single bank is limited and, in fact, has not increased over the last decade. For reference, if we take relatively new hardware, memory bandwidth per channel is around 20GB/s, while the bandwidth of a single bank assuming 48ns delay [21] for row conflicts is 1.33GB/s. The implications of this are that a bad mapping or unlucky accesses to the same bank can cause the performance of the memory to drop below advertised or acceptable levels.

2.3 Remote Direct Memory Access (RDMA)

RDMA is a technology deployed at the end-points in a computer network which aims to offload the network stack and to provide faster remote memory access than conventional networks. It originates from high-performance computing and it is widely adopted by cloud providers like Microsoft and Oracle. There are two main RDMA architectures – Infiniband and RoCE (RDMA over Converged Ethernet). Infiniband is designed to support RDMA with its own hardware and protocols, while RoCE enables RDMA over an Ethernet network. This means that RoCE can run on existing Ethernet infrastructure.

RDMA implements one-sided read/write transfer operations. They are handled by RDMA network interface cards (RNICs or RDMA NICs). RNICs are specialized hardware components, required to deploy RDMA. RDMA operations are performed asynchronously allowing the sender to continue its operation without waiting for the operation's completion. RDMA operations bypass the CPU of the receiver which drastically reduces its utilization.

The comparison of RDMA and conventional networks can be seen in Figure 2.2. In conventional TCP/IP networks, the application-level data is wrapped on the CPU of the sender, which involves multiple copy operations. Only then, the data is sent to the NIC and over the network, before it has to be unwrapped by the receiver's CPU. On the other hand, RDMA transfers data from the virtual memory of the sender application directly to the RNIC and over the network, without intermediate copy operation that take up valuable CPU cycles.

These properties allow RDMA to offer a low latency and a high throughput network as well as to boost the performance of all applications by reducing the CPU overhead of the network.

Before a memory region can be accessed via RDMA, it needs to first be registered with appropriate permissions by the application that owns it. After that, the registered memory cannot be swapped out of the memory to the disk and it can be accessed by a remote application similar to how that application would access its own address space – by including the virtual address and the size of the request in an RDMA packet.

At the destination side, an RDMA read packet is translated into a sequence of memory requests. This commences a DMA read transaction that is processed by the CPU root

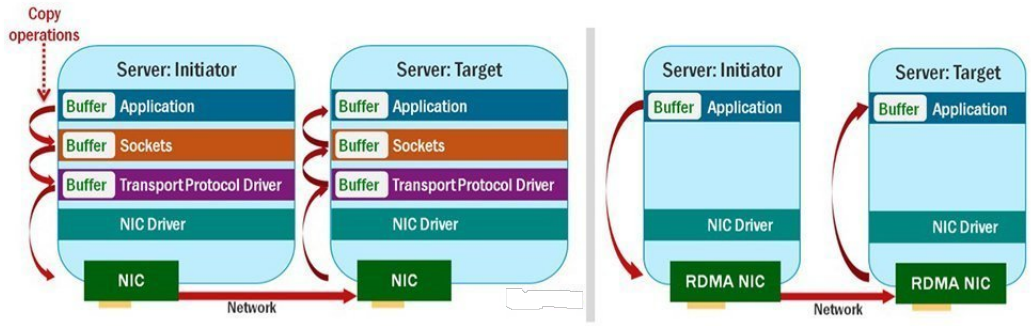


Figure 2.2: TCP communication on the left compared to RDMA communication on the right. [15]

complex. The root complex then forwards them to the memory controller which sends them to the corresponding bank's queue. The memory accessed by RDMA is not cached so the requests always go to the main memory.

The RDMA protocols do not guarantee strong security properties. RDMA implements keys associated with each memory region, which need to be passed when the memory region is accessed. Keys function like a weak authentication measure. Other than that, RDMA does not provide any encryption on its own yet, although there are attempts to do this. Furthermore, by bypassing the destination CPU and the OS, RDMA exposes the details of the low-level memory interfaces such as the memory inter-leaving scheme and the virtual address space in the destination machine.

2.4 Related Work

In the related work for covert channels, we also have to briefly consider side channels. Side channels are attacks where an attacker is able to extract confidential information from a victim without permission. Attackers can obtain the secret through the victim's usage of a resource that both parties share. One use case for covert channels is to transmit the information retrieved via a side channel. The two types of attacks should be considered together in the related work. For example, the timing difference in a shared resource which allows one type of attack can be used to construct the other one.

There are multiple cache covert and side channels, which exploit the timing difference between cache hits and cache misses. PRIME+PROBE cache attacks have been developed which can target a particular cache set. Hyperthreading [16] has been exploited to establish a side as well as a covert channel between processes running on the same physical core [24]. Cross-core attacks [18, 14, 8] exploit the shared last level cache which allows an eviction from it to result in an eviction in the L1 cache of a target process. This relies on the successful reverse-engineering of the placement function in the last level cache [17, 36]. FLUSH+RELOAD cache attacks [10, 35] can exactly identify a cache line, but they require shared memory which is normally disabled in a cloud setting.

One reason for the prevalence of cache covert and side channels is that they are the

first level of memory that is accessed for each memory request, which makes it easy to reason about their timing behaviour. What is more, caches are very well-understood, which helps to reverse-engineer any undocumented implementation. Usually, only few accesses to a cache are required to achieve a desired result, such as evicting a particular cache line or set. This, and the short cache access time, are the reasons for the relatively high bandwidth of cache attacks.

Memory has also been a prime target for covert and side channels attacks. Several attacks have exploited memory deduplication [29, 3, 33]. This is a technique employed in a virtualized setting to reduce the physical memory used by sharing same-content memory pages. The memory bus has also been used for covert channel communication in a virtualized setting [32]. The Drama covert channel [25] is very closely related to the Bankrupt attack because it targets memory banks as well. It fully reverse-engineers the processor's mapping from physical address to bank in memory and uses this information to retrieve addresses which map to the same bank but a different row. It then exploits the timing difference between row hits and row conflicts to establish covert channel communication between parties which share the same memory module.

While most of the mentioned attacks rely on the attacker's capability to collocate their malicious software on the same CPU, multiple covert and side channels have been discovered that do not require collocation on the same physical machine. NetCAT [12] is a side channel which is able to retrieve SSH keystrokes over the network. It effectively uses Intel DDIO [6] to mount a cache attack from the RDMA network. Pythia [30] is a another side channel over the network which allows one RDMA client to find out what addresses another RDMA client has accessed from an RDMA server. To do this, it reverse engineers the memory architecture of the server's RNIC and exploits it to monitor what memory the victim accesses. Both of these attacks leverage the low-level information exposed by RDMA about the remote virtual memory.

The Bankrupt attack is related to the mentioned attacks in that it enables covert channel communication between machines in the cloud. Similarly to the Drama attack, it uses a timing channel in the memory bank to transfer information. Nevertheless, differently from the local Drama covert channel, Bankrupt is a remote covert channel. To the best of our knowledge, there does not exist a covert channel over an RDMA network which enables the reliable transmission of messages from a sender to a receiver via a timing channel in the memory of a third server that both parties have access to.

Chapter 3

Methodology

In this chapter, we describe the threat model for the Bankrupt attack – we discuss the assumptions we make for the system and the tools at the attacker's disposal. We also motivate our philosophy for the attack's design and summarize the specifications of our experimental platforms.

3.1 Threat Model

Our attack assumes that the sender and the receiver programs run on separate machines within an RDMA-connected cluster. They are not allowed to communicate with each other legitimately through the network. We assume that privilege escalation is not possible and the two communicating parties have normal user privileges. As a result, they cannot alter any of the firewall policies that are in place. Any of the communicating parties may run in a virtual machine, and they do not have any means to compromise the job placement algorithm to share the same CPU core or even the same CPU to use one of the known local covert channels. None of the two parties have means to observe network traffic as otherwise the receiver could sniff the network to decode the covert communication.

In this setting, to establish a covert communication channel, the sender and the receiver both need to be able to communicate with a third machine (or set of machines) over RDMA. The intermediary machine can be non-malicious with the sole requirement that the sender and the receiver can access its memory with RDMA read (or write) operations. In order to identify a machine and a bank for the covert channel communication, the sender and the receiver need to issue probes. For example, the sender can send out an agreed preamble message to multiple banks in multiple machines that it can connect to. At the same time, the receiver can issue probes to multiple banks as well in hopes of detecting the preamble message. Once a bank has been found in this way, the transfer of data can begin. Although ideally this bank agreement will happen fast, due to the fact that the channel is undetected, the sender and the receiver are not in a rush to find a bank to send the information.

This is a realistic setting in modern private cloud infrastructures. Services are placed

	UoE cluster	**CloudLab**
CPU	Xeon E5-2630v4 @2.20GHz (Broadwell)	Xeon E5-2450 @2.1GHz (Sandy Bridge)
RAM	4x16GB, DDR4, 2400MHz	16GB, DDR3, 1600MHz
NIC	Mellanox MT27700 FDR CX-4	Mellanox MX354A dual-port FDR CX-3
Core switches	N/A	2 Mellanox SX6036G
ToR switches	Mellanox SX6012	7 Mellanox SX6036G
OS	Ubuntu 18.04	Ubuntu 18.04
Kernel	4.15.0-58-generic	4.15.0-55-generic
Nodes	6	192

Table 3.1: Specifications of demonstration platforms.

on virtual machines in order to ensure better isolation and resource utilization. They rarely have special privileges which allow them to alter firewall settings or network policies in the data center and it is normally not possible to escape the virtual machine sandbox and to acquire root privileges on the host machine.

3.2 Experimental Platforms

We develop the covert channel on a private cluster at the University of Edinburgh. Our cluster offers high customization and isolation, which makes reasoning about the channel behaviour easier. It also allows us to modulate the level of congestion in the cluster's network and the workload on any one machine at any time. This helps us evaluate the functionality and the performance of the channel under different cluster loads.

In order to demonstrate the practical merit of the channel, we also evaluate the channel on CloudLab [1] – an infrastructure for cloud computing research. CloudLab offers high customization as well, but, differently from our private cluster, it has other active users who run various workloads. This is a more realistic representation of the cloud environment which our covert channel attack targets.

The specifications of the two platforms are presented in Table 3.1. The processors on both platforms are currently in use in commercial clouds [2]. Although the Sandy Bridge microarchitecture is comparatively older than Broadwell, it is important to be able to demonstrate the covert channel on different platforms with no changes. This heightens the potential impact of the attack and it can show a fundamental security weakness in multiple generations of Intel processors and various memory technologies as well.

We began the development of the channel with multiple kernel options in place such as 1GB Hugepages, disabling Turbo Boost [4] and Simultaneous Multithreading (SMT) [31]. The aim was to control the environment in which the channel is executed in order to more easily form our expectations and rule out different reasons for the channel's behaviour which made debugging easier. Gradually, we began dropping these assump-

tions to make sure that the channel does not rely on any of them, which would limit its practical usefulness. The only kernel parameter we leave is 1GB Hugepages. This is a common optimization in RDMA-based systems since even a low miss ratio in the virtual-to-physical mappings cache results in a severe performance degradation [7].

Chapter 4

The Bankrupt Covert Channel

The purpose of this chapter is to present the design and implementation of the Bankrupt covert channel. We begin by describing how the memory organization into banks and the high bandwidth of RDMA networks provide an opportunity for the attacker to construct a covert channel. Then, we present an overview of the channel and explain what the resulting challenges are and how we approach them. Finally, we delve into the decoding process and discuss possible mitigations for the channel.

4.1 Opportunity for the Attacker

If an attacker wanted to use main memory blindly as a shared resource to establish a covert channel, they would normally need a way to create a timing difference in the response time from *all* banks so that the timing difference will be noticed by *any* memory request. Because the total memory bandwidth of modern systems is high ($\sim$20GB/s per channel), this would require putting an enormous load on the memory, which has significant drawbacks. It can easily be detected and it might not be possible if the attacker has a limited number of cores at their disposal.

However, the attacker can exploit the segmentation of memory into banks to issue requests targeted at a single bank. In this way, a much smaller load can cause an explosion in the latency of accesses to this bank, which can be used to establish a timing channel in that particular bank. Such smaller load is also harder to detect and it is less likely to impact other applications in a way that the global performance of the system degrades so that the presence of the attack can be noticed.

For a general remote covert channel where the sender and the receiver are not on the same physical machine, the sender would need to create a timing channel in a bank *from the network*. Conventional TCP/IP networks are ill-fitting for this purpose, as they do not reveal information such as the address space to allow the attacker to direct requests to a single bank. Furthermore, TCP/IP has major overheads which makes it a lot harder for the receiver in a covert channel to decide if a delay is being caused by the sender or somewhere along the network stack.

RDMA resolves the problems caused by TCP. It exposes the details of the low-level

memory interfaces such as the memory interleaving scheme in the destination machine, enabling the sender machine to target RDMA accesses to a particular bank on the receiver. What is more, nowadays, private clouds are equipped with 100 Gbps [26] and 200 Gbps RDMA networks [22]. Such bandwidths are approximately 10 times larger than the bandwidth of a single bank (1.33GB/s as noted in §2.1). This allows the attacker to completely overflow the queue of a memory bank remotely. Furthermore, the predictability of RDMA networks [11] increases the correlation between memory congestion in the memory of the destination machine and delay in the RDMA operation as a whole, making it easier for the receiver in a covert channel to decode the signal. Finally, because the receiver's processor is oblivious to one-sided RDMA operations, a covert channel attack which uses one-sided RDMA operations becomes hard to detect. These are ideal conditions for the establishment of a covert channel.

4.2 Overview

The Bankrupt attack is a covert channel which enables the communication between a sender and a receiver in an RDMA cluster which are prohibited to transfer information between each other over a legitimate channel. Our attack creates a timing channel in one of the memory banks of a third machine in the same cluster that both the sender and the receiver can communicate with.

The covert channel has 3 distinct participants: the sender (S), the receiver (R), and the Intermediary (I). These can be any arbitrary machines in the cluster as long as both the sender and the receiver are connected to the intermediary over RDMA. This makes the attacker's surface very wide. Figure 4.1 illustrates the role of each of them and operation of the channel.

The intermediary runs a service that allows the sender and the receiver to access some of its memory regions via RDMA. The intermediary does not have to be malicious.
The sender identifies a set of addresses in the memory region that it has access to on the intermediary machine. These addresses map to the same bank in the intermediary. Then, the sender can modulate the latency of the intermediary's bank for a short time by sending a burst of RDMA reads to the intermediary. It sends a burst to the intermediary to transmit a 1 over the channel, and it does not send anything to transmit a 0.
The receiver also identifies a set of addresses that map to the same bank in the intermediary. The receiver then sends RDMA reads to the identified addresses at a constant rate and records the round trip times. In this way, it builds a trace of addresses which it can then decode to retrieve the sender's message.

We present the most challenging setting for the covert channel. In this case, all three participants are located on separate machines in the RDMA cluster. This is also the most dangerous scenario because it allows the covert communication between arbitrary nodes in the cluster. However, the attack can work even if any of the three parties are collocated on the same machine. For example, collocating the sender and the receiver does not impact the attack in any way. Collocating the receiver and the intermediary can actually be beneficial for the attack. The reason for this is that the probes from the receiver to the intermediary will not have to go through the switch in the network.

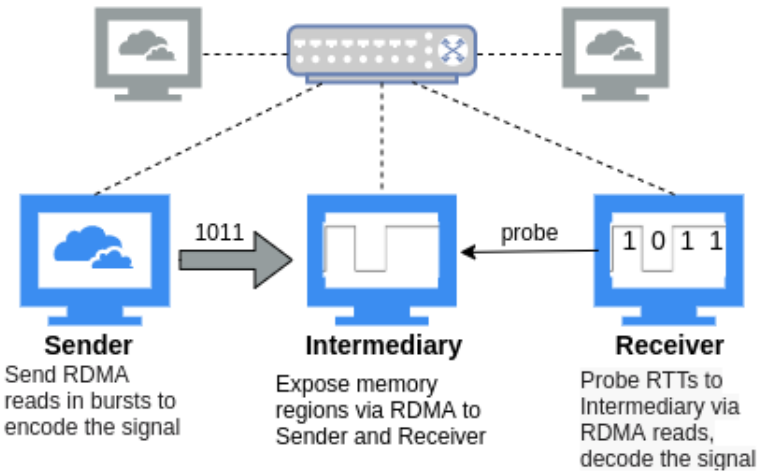


Figure 4.1: Overview of the Bankrupt covert channel. The figure shows the functions of the sender, the intermediary and the receiver as well as the presence of other machines in the cluster.

This will eliminate some of the variability in the round trip time due to scheduling in the switch, and will make the decoding process more robust. Collocating all three participants will increase the bandwidth of the channel due to the shorter round trip time of the bursts. Thus, the Bankrupt attack can be used as a local covert channel as well.

4.3 Challenges

Following the high level description of the covert channel, we can identify multiple challenges in the design and implementation. In this section, we explain how the sender and the receiver identify addresses which map to the same bank in the intermediary. We also describe how to minimize the memory footprint of the channel and elaborate on the requirements for the sender's rate and the receiver's probe interval.

4.3.1 Finding Addresses that Map to the Same Bank

The first necessary step to construct the covert channel is for the sender and the receiver to identify a set of addresses which map to the same bank in the intermediary's memory. To start with, the sender and the receiver request a region of memory from the intermediary and they are provided with the starting virtual address and the size of the region. The procedure for identifying a set of addresses is the same for both parties so we show it from the sender's perspective.

As explained in the Background section, the mapping from *physical* addresses to banks is a function of some bits in the physical address and Intel does not make this information public. The Drama tool [25] has been used to fully reverse engineer this memory mapping in a number of Intel processors. It alternately accesses pairs of memory locations and leverages the increased latency resulting from row conflicts to construct sets

of addresses which map to the same bank, but different rows. In the next stage, the mapping function is determined based on these sets.

The memory configurations of our experimental platforms are different from the ones presented in Drama so we cannot readily use the information in the paper. This is also undesirable as we would like to make the covert channel generalize to any memory configuration. The sender might not have knowledge of the memory configuration of the intermediary in a real cloud environment. What is more, the Drama approach assumes the host machine is being reverse engineered. Using the same approach to reverse engineer the intermediary's memory from the sender via RDMA might not work as the row conflict timing differences might not be visible from the network.

Nevertheless, it is important to note that for our purposes we do not need to know the full mapping from addresses to memory banks. It is sufficient to be able to identify a group of addresses which map to the same bank for *one bank*. We still use the knowledge from Drama that in many Intel microarchitectures the mapping from physical address to the bank is determined by a function of some lower bits.

Because the processor maps physical addresses to banks, we need knowledge of the mapping from virtual to physical addresses. This requires access to the page tables and it is normally not available without any privileges. However, we can utilize 1GB Hugepages so that the 30 least significant bits of the virtual and physical address overlap. In all of the presented configurations in Drama, knowing the 30 least significant bits is sufficient to reverse engineer the bank hash function.

Because we are not able to use the Drama results directly, we take an experimental approach to identifying addresses that map to the same bank. It relies on the realization that if we fix the bits in the virtual address from the least significant bit to the most significant which is used to determine the bank (inclusive), then all addresses where these bits are the same will map to the same bank. This is because the output of the hashing function will be the same. Our approach has two important benefits. First, it removes the need for knowing what exactly the function is, even though it has been shown that it is usually a linear function of some bits (XOR). In addition, it becomes irrelevant which exact bits the function takes as inputs, because we fix all the bits that determine the bank.

Still, fixing all 30 known bits of the physical address has a major drawback. It means that the sender can only find one address per 1GB of memory that it has access to in the intermediary. Because for the attack we need tens of addresses to create a visible timing channel, this would make the memory requirement of the attack tremendous and its practicability limited.

As a result, we begin fixing bits 1 by 1 from the least significant bit to the more significant bits. We design a benchmark to continuously access addresses which share a number of their least significant bits and monitor the memory bandwidth during the execution. As we fix more bits, the overall memory bandwidth decreases, because the load spreads out on fewer and fewer banks. Knowing that the maximum bandwidth per memory bank on our private cluster is around 1.3GB/s, we fix bits until we see the memory bandwidth saturate to this number and stop decreasing even if we fix more

bits. The bandwidth stops decreasing when we add more bits, because it is already concentrated on a single bank that is getting fully utilized.

On our private cluster, we find that fixing the first 27 bits of the address is enough to direct traffic to a single bank. The configuration can be seen in Figure 4.2. This significantly reduces the memory required by the channel. With this configuration, we can have 8 addresses per 1GB Hugepage, as there are 3 remaining bits of the physical address that we know. Furthermore, the same number of fixed bits works in our CloudLab experiments, which gives us confidence in the generalization of our approach to multiple Intel processors.

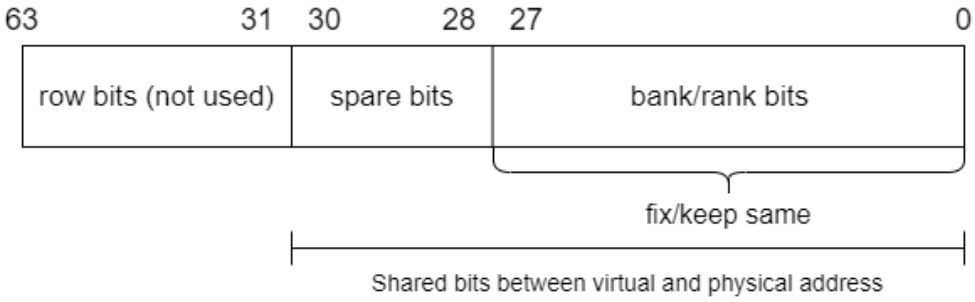


Figure 4.2: Role of virtual address bits in identifying addresses which map to the same bank.

4.3.2 Tuning Sender and Receiver Rate

The sender submits a burst of requests to transmit a 1 to the receiver and does not send anything to transmit a 0. In order to maintain a robust signal, the sender needs to ”keep quiet” for the same period that it takes for a burst. For this reason, the sender tunes the quiet interval before it starts communication by measuring how long it takes for a burst to complete. The sender then keeps quiet for that period of time to send a 0. The sender can recompute this interval periodically to take into consideration the changing load in the cluster.

The receiver needs to probe the intermediary’s bank in order to receive the signal. The frequency of probing is important because it directly impacts the receiver’s ability to reconstruct the signal. Not probing frequently enough means that the receiver can miss a burst altogether. Also, the receiver can get unlucky and see low latency while the sender is actually sending a burst. In this case, the receiver will record a 0, whereas it is being sent a 1. Such temporary inconsistencies can happen due to queuing in the NIC or in the bank. These possibilities are shown in Figures 4.3 and 4.4. As a result, we would like the receiver to probe as frequently as possible.

On the other hand, issuing requests too frequently might cause queuing of the receiver’s probes in the intermediary’s bank. This increases the round trip time of the probes and might deceive the receiver into seeing larger latencies. **Empirically, we find that 500ns was a reasonable probing interval to allow the robust reconstruction of the signal without the receiver overwhelming the intermediary.** This frequency is motivated by the fact that in our cluster the round trip time of an RDMA read without

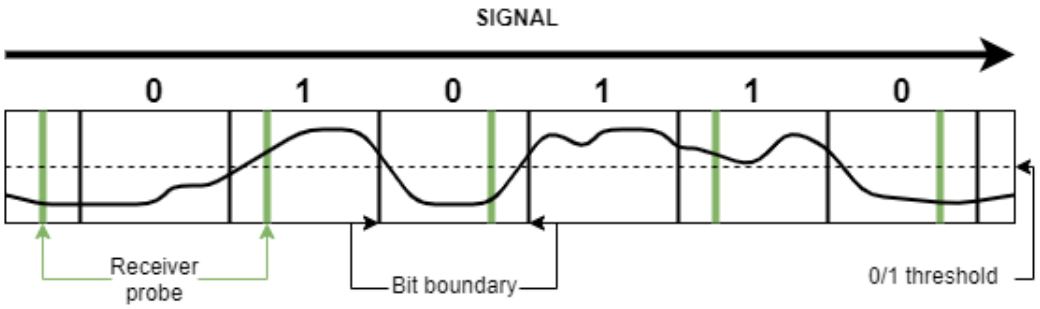


Figure 4.3: Receiver probe frequency too large. The receiver misses out on the first and the fourth bit of the signal.

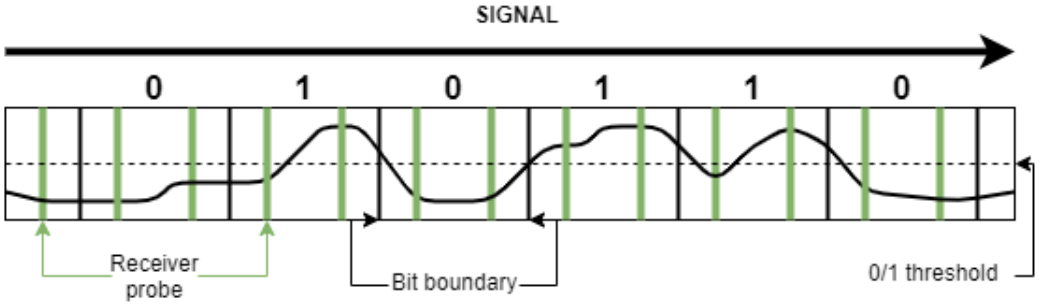


Figure 4.4: Receiver is unlucky in one of the probes for the 2nd and the 5th bits, but the other probe is able to detect the burst.

any load in the network is around 2 microseconds, which results in a minimum of 3 probes per round trip time.

4.4 Decoding

Once the receiver has finished probing the intermediary, it has to decode the signal from a trace of recorded latencies. Every bit in the signal is represented by a sequence of consecutive probes in the trace. In order to retrieve the message, the receiver has two tasks:

1. To cut up the list of probes into non-overlapping equal chunks of consecutive probes where each chunk encodes a bit.
2. To identify whether each chunk encodes a 0 or a 1.

These tasks are very closely related. If we know the chunk boundaries (Task 1), we can decode each chunk (Task 2) as follows:

1. The receiver sends a number of consecutive probes to the intermediary, while the sender is *not* sending anything. In this way, the receiver records many round-trips while the intermediary's bank is unloaded. This is the round trip that the receiver observes when the sender is not sending a burst, so it is the round trip for a 0.

2. The receiver calculates the 95th percentile of the obtained latencies. This serves as the threshold to determine if each chunk is a 0 or a 1.

3. The receiver compares the 80th percentile of the latencies in each chunk it is decoding with the threshold. If it is smaller than the threshold, then the chunk is a 0. Otherwise, it is 1.

This is an experimental approach that relies on the fact that the latencies in the RDMA network are very predictable and the percentiles have small fluctuations. As a result, the fact that the 80th percentile of a chunk is bigger than the 95th percentile of the round trips for a 0 gives us strong confidence that the chunk is a 1.

Next, we look at how to identify the boundaries of each chunk (Task 1). Task 1 can be divided in two subtasks: finding out the length of each chunk in terms of probes, and putting the chunk boundaries in the signal. We include Figure 4.5 to aid with the understanding of this task. The text and the figure are best read together.

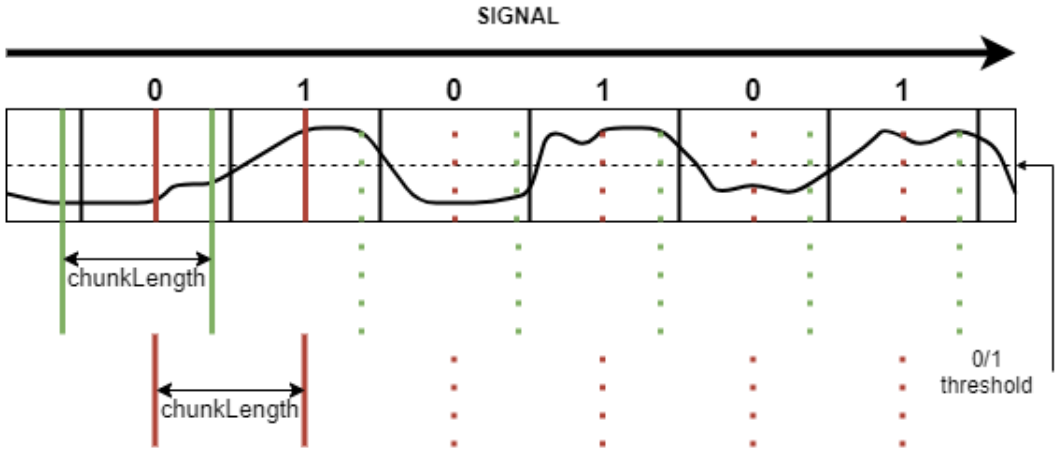


Figure 4.5: Identifying chunk boundaries in the signal. The figure shows the signal that the receiver has obtained by probing. The black vertical lines show the ”true” chunk boundaries in the signal. The green vertical lines show possible boundaries which identify the signal without any errors. The red vertical lines show a split which is very different from the true boundaries. Indeed, we can see that half of one bit and half of another make up each red chunk. The decoder only needs to identify the first two boundaries. These are marked with non-dotted lines. The decoder can then place the other boundaries, plotted as dotted lines, because it knows the length of each chunk.

The receiver knows the burst size used by the sender as it is agreed upon between the sender and the receiver before the covert communication begins. The receiver also knows the time interval between any two consecutive probes – this is the *probeInterval* that the receiver uses when probing the intermediary (500ns in our experiments, as noted previously). We discuss the probe interval in Section §4.3.2.

At this point, the receiver can issue a burst to the intermediary with the same burst size used by the sender. This tells the receiver how long each burst takes and, consequently, how long each chunk lasts for in the signal in nanoseconds – we call this *period*. The

receiver can issue multiple such requests and average over them or take the median to be more resistant to outlier latencies.

The receiver can then calculate the length of each chunk *in number of probes* as follows:

$$chunkLength[probes] = \frac{period[ns]}{probeInterval[ns]} \tag{4.1}$$

Once the receiver knows the length of each chunk, it needs to determine where to put the chunk boundaries. Currently, the channel uses a preamble message and we manually determine the first chunk boundary in the preamble so that the preamble is recognized without errors. From then on, the receiver can split the signal into chunks of *chunkLength*, decode each chunk, and reconstruct the payload. We use a 32 bit payload of alternating 1's and 0's. We find that this preamble length allows us to tune the decoder to minimize the error rate (Chapter 5). What is more, the preamble has the same number of 1's as 0's, which puts equal weight on reconstructing both 1 and 0 chunks correctly.

4.5 Mitigation

We propose several mitigations which make it more difficult to mount the attack. The Bankrupt covert channel leverages a range of technologies and design decisions which are optimized for performance – RDMA, the memory organization into banks, the memory controller's mapping function, and 1GB pages. For this reason, it is difficult to mitigate the attack without incurring at least some performance degradation.

The most crude method of mitigating the attack is to disable RDMA or go back to a previous memory technology which does not have banks. This is impossible as tremendous amounts of resources have been invested in these technologies and they have accounted for immense performance gains. The most feasible technique is disabling 1GB pages, but for large scale applications which run in data centers, even Hugepages have become crucial. As a result, we need to consider measures to enhance the existing technology in order to mitigate the Bankrupt attack.

The first mitigation is to introduce *strong inconsistent noise* into the RDMA network. Strong noise would make the decoding job of the receiver more complicated as the round-trips observed will be caused by the strong noise present in the cluster. Inconsistent noise can also induce large variability in the time it takes for a burst. This makes the sender difficult to tune to keep bit length in the signal constant.

This mitigation is easy to implement because it does not require any changes to the existing technology. However, it makes the network slower and more unpredictable, negatively impacting other non-malicious applications in the cluster. The next set of mitigations we consider require more work to implement but they hinder performance less severely.

One possible mitigation is to alter the memory controller's mapping from physical addresses to banks so that it includes higher order bits, such as bits from 30 and above.

In this case, even if we use 1GB pages, we can no longer guarantee that bits above 30 will be same in the virtual and the physical address. This makes our approach (Section 4.3.1) of fixing bits to find addresses which map to the same bank unusable. This is because fixing these bits in the virtual address does not mean that they will be the same in the physical address. This makes it possible for two virtual addresses, where the bits which determine the bank are the same, to be in a different bank.

This mitigation has a performance detriment as well. Manufacturers have not used more significant bits to determine the bank, because these bits have a lower entropy. This can lead to a poor spread of the overall memory load to the banks. In this way, some banks get overutilized, while others remain underutilized.

Furthermore, we propose to make RDMA Network Interface Cards (RNICs) aware of the memory controller's mapping from physical addresses to banks. This allows the RNIC to implement an intrusion detection system (IDS) to flag suspicious network behaviour. Our solution is inspired by software defined networking (SDN), where the notion of a *flow* is used to group traffic with the same origin and destination, and using the same application. In the same spirit, the IDS on the destination machine can be used to group RDMA requests coming from the same source and going to the same bank, and look for anything suspicious.

For example, one type of unusual behaviour is when an RNIC receives many requests *from the same source different addresses which map to the same bank*. This is the kind of behaviour that the Bankrupt attack has and it is undoubtedly unusual, because ordinary applications normally generate approximately equal load across all memory banks. The intrusion detection system can raise an alarm to the cloud administrators, block the traffic altogether or stall requests so that they do not cause a burst in the memory bank.

Making the RNICs flow-aware is an attractive mitigation, because it has next to no adverse effect on the performance of non-malicious applications. However, implementing it requires additional logic on the RNIC, which will make these components even more expensive than they already are. For example, a 100 Gbps RNIC can cost close to £1000. Last but not least, IDSs are notorious for being difficult to tune so that the number of false alarms is low, while detecting all intrusions.

Chapter 5

Evaluation

In this chapter we present and discuss the results of the experiments we carry out to evaluate the channel's performance. We use two experimental platforms – our private cluster as well as the CloudLab Apt cluster. We give their specifications in Table 3.1.

First, we provide a methodology section where we summarize the characteristics common to all of the experiments. Then, we explore the channel's bandwidth and its robustness in isolation and under load on the two experimental platforms.

5.1 Methodology

We use our cluster to carry out all of the studies concerning the channel's behaviour in isolation and under load because we can moderate the overall network and memory traffic in the cluster at any given time. Hence, we can show results and reliably reason about the channel's behaviour under different settings. We use the CloudLab cluster to demonstrate the channel in a realistic cloud setting. The assumption is that other users of the cloud already apply load on the network.

All of the experiments that we carry out share the following characteristics:

- The sender and the receiver use RDMA Read operations to create bursts or probe the intermediary, respectively.
- The size of each RDMA Read is 64 Bytes.
- We batch the sender's requests in order to make sure that they arrive at the intermediary as a burst, momentarily creating a queue in the memory bank. RDMA limits the batch size to 16. For bursts which are larger than 16, we send multiple back-to-back batches from different QPs, but from the same thread.
- The sender uses at most 32 unique addresses to the same bank to create a burst. For bursts that are larger than 32, the sender reuses the addresses. We find empirically that using more unique addresses does not impact the latencies observed by the receiver. This is likely due to the fact that the memory bank is not able to

reorder more than 32 requests. For bursts which are smaller than 32, all of the requests are to unique addresses.

This reduces the memory requirement for the channel and makes it scale very well with bigger burst sizes. In Section §4.3.1 we show that we can identify 8 addresses that go to the same bank per 1GB page. This means that the sender requires at most 4GB from the intermediary, regardless of the burst size that it is planning to use.

- The message exchanged between the sender and the receiver is a repeating sequence of "11001010". In Section §4.2, we mention that the sender issues bursts to transmit a 1, and it remains quiet to send a 0. Our message format contains two consecutive 1s, two consecutive 0s, a 1 followed by a 0, and a 0 followed by a 1. Consequently, it allows us to observe the channel's performance for various burst/quiet time combinations: two bursts in a row, two quiet periods in a row, a burst after a quiet period, and a quiet period after a burst.
- We perform the decoding using the procedure described in Section §4.4 and calculate the error rate as the fraction of misdecoded bits among the first 200 bits of the payload following the preamble.

5.2 Performance in Isolation

The first experiments that we carry out aim to quantify the throughput and the robustness of the Bankrupt covert channel in an isolated environment. The burst size used by the sender impacts both the bandwidth and the robustness of the channel. What is more, there is a trade-off between the two. Bigger bursts take longer which increases the time it takes to transmit a bit and reduces the throughput of the channel. However, bigger burst also create a longer queue in the intermediary's bank. This allows the receiver to reconstruct the signal with fewer error. In this section, we show and discuss the performance of the channel for different burst sizes.

We execute our experiments in an isolated environment on our private cluster. We stop all the other processes in the cluster apart from the sender, receiver, and intermediary processes. This allows us to limit the number of factors that can affect the channel behaviour and to attribute the change in performance to the burst size alone.

Figure 5.1 shows parts of the signal recorded by the receiver for different burst sizes. We use similar figures to show the receiver's view of the signal throughout the chapter. The figures share the following characteristics:

- They show the latency of the receiver's probes to the intermediary. In agreement with Section §4.3.2, the receiver issues probes every 500ns.
- The horizontal X axis is the time in μs.
- The vertical Y axis is the latency of the probes in μs.
- Each red dot represents a probe. Given the probing frequency, there are 2 probes or dots every microsecond.

- The blue curve connects the probes to create a ”waveform” of the signal.
- We keep the limits of the X axis the same where possible. This is so that the reader can see the impact of the burst size on the bandwidth of the channel. For smaller burst sizes, we expect to see more bursts in the same time than for larger bursts
- We keep the limits of the Y axis the same where possible. This is so that the reader can see the impact of the burst size on the robustness of the signal. For bigger burst sizes, we expect bursts to be easier to identify.
- We use vertical lines to indicate the bit boundaries in the signal.

First, we note that increasing the burst size also increases the overall latency that the receiver observes during bursts. This is because larger bursts create a bigger queue in the intermediary's bank, which causes the receiver's requests to wait for longer. We can see that for burst size of 128, the latency for around half the bursts goes above $3.5\mu s$. On the other hand, the latency rarely goes above $3\mu s$ for burst sizes 64 and 32.

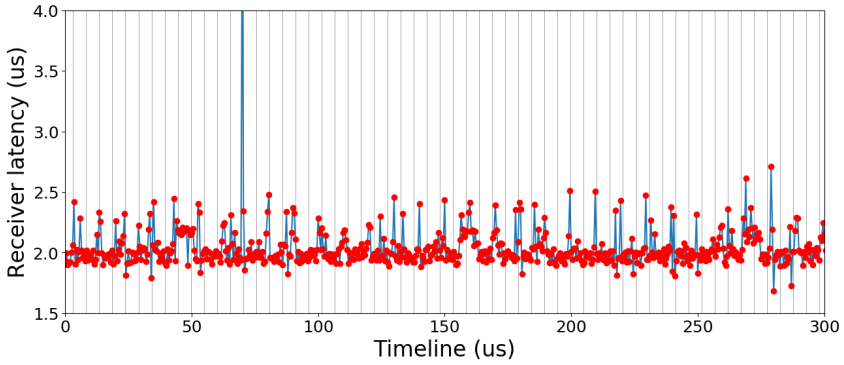
Furthermore, we can observe from Figure 5.1 that bigger bursts are easier to see, as they take longer to process by the intermediary's bank. Longer bursts also allow the receiver to have more probes per burst, which makes the signal more robust. Bursts of 128 are easy to see, while burst of size 16 can hardly be told apart from quiet periods.

In Table 5.1, we show the throughput and the error rate, as calculated by the decoding procedure presented in Section §4.4, for different burst sizes. We immediately confirm our observation that bursts become harder to identify as we decrease the burst size, because the error goes up. Moreover, we see that the throughput increases as we lower the burst size, which we expected because of Figure 5.1.

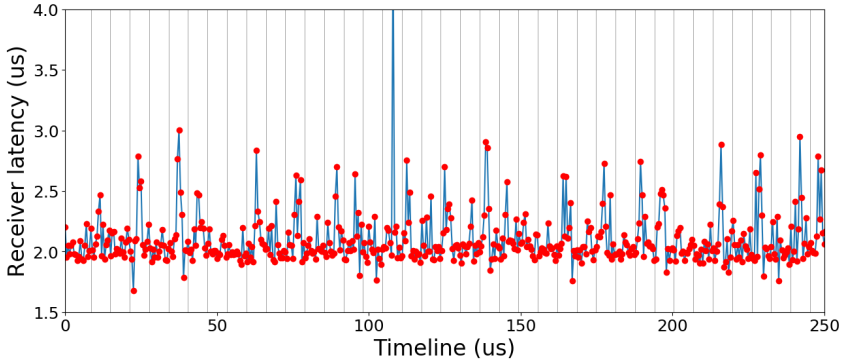
Burst Size	Period (μs)	Throughput (Kbps)	Error rate (%)
128	11.13	89.85	1.5%
64	7.9	126.58	2%
32	6.3	158.73	5%
16	5.17	193.42	49%

Table 5.1: Throughput and error rate of the Bankrupt channel in an isolated environment. The period is the round-trip time per burst, and respectively, the time the sender remains quiet.

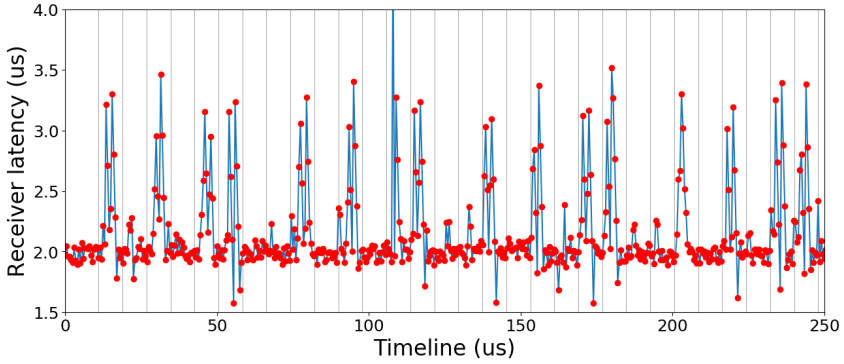
Last but not least, we see a drastic increase in the error for burst size of 16 compared to burst size of 32. We could already tell from Figure 5.1 that for burst size 16 the bursts become almost indistinguishable from times where the sender is not sending anything. Burst of size 16 cause only 1 or 2 probes to have a higher latency for each burst, which is not sufficient to identify a 1. As a result most chunks are decoded as 0. Although the raw throughput of the channel for burst size 16 is the highest, the enormous error rate is intolerable. **We achieve the best throughput of 158.73Kbps with error rate of 5% for burst size of 32.** It is possible to fine-tune the burst size further between 32 and 16 to achieve higher throughput and lower error.



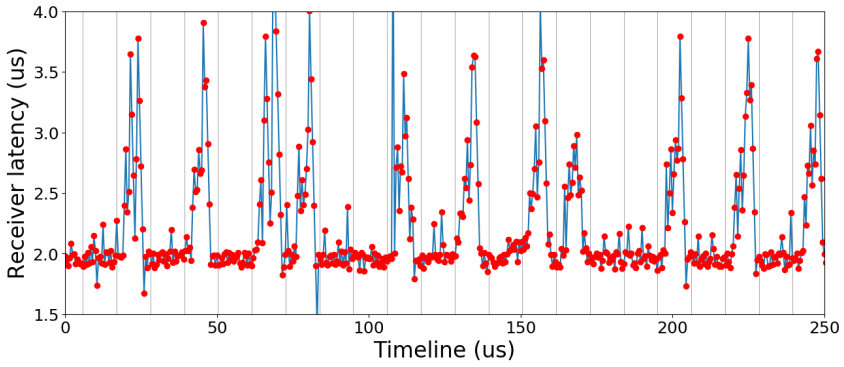
(a) Burst 16



(b) Burst 32



(c) Burst 64



(d) Burst 128

Figure 5.1: Parts of the Bankrupt channel signal for different burst sizes in an isolated environment on our private cluster.

Although we present the throughput and the error rate of the Bankrupt channel in this chapter, for a fully working mechanism in practice, we need to perform error checking or error correction. We do not implement this functionality but discuss it in the future work (Chapter 6.1).

5.3 Performance Under Load

An isolated environment like the one we confined the channel in in the previous section is not achievable in a real cloud. We cannot force the other applications to stop, nor can we expect that no applications will be running at any given time. This is why it is important to evaluate our channel under a load that we are likely encounter in a real data center. It is important to point out that this does not mean that we have to stress the system, as it has been shown that machines in clouds are often underutilized [11].

5.3.1 Local Intermediary Load

We evaluate the channel in the case where there are other applications running locally on the intermediary machine which generate traffic in the intermediary's memory. In this setting, it is possible that the memory requests of the other local processes affect the receiver's observed latency negatively. For example, the other applications might cause congestion in the memory while the sender is not sending anything, leading the receiver to believe that a 1 is being transmitted. Furthermore, memory requests of other applications can cause the bursts of the sender to queue for longer, potentially deceiving the receiver that two consecutive 1s are being sent instead of just one.

In order to model the local load in the intermediary's memory, we launch 16 mcf [9] benchmarks on 16 separate threads on the intermediary machine. We stagger the starting times of each benchmark by 1 second to ensure that not all benchmarks are in the same stage. The benchmarks together generate variable load on the intermediary's memory ranging from 2GB/s to 8GB/s.

We find that the mcf benchmarks do not have any significant effect on the performance of the channel. Indeed, the error remains the same and the throughput differs by 0.15Kbps for burst size 32, which is likely due to the small natural variability in the latency of RDMA requests. The reason why the channel is not affected by the benchmarks is that each individual bank is affected very lightly by the load created by the benchmarks, which spreads out to all banks in the system. In practice, due to the way in which DRAM is organized, it is difficult to find a application or a benchmark which will put high load on a single bank, without stressing the memory of the system.

5.3.2 Network Load

In this section, we examine the performance of the channel while the intermediary is receiving network traffic from other applications running in the cluster. In a real cloud environment, the intermediary will likely be collocated with other applications that use the RDMA network as well. Because of this, it is important to evaluate the channel's

Burst Size	Throughput (Kbps)	Error rate (%)
128	81.33	7.5%
64	111.11	14.5%

Table 5.2: Throughput and error rate of the Bankrupt channel in the presence of network load.

performance in such a setting. The network traffic generated at the intermediary's NIC can cause problems for both the sender and the receiver.

The receiver can begin to experience inconsistent round-trip times, even when the sender is not sending anything, due to the receiver's probes having to queue in the intermediary NIC. Such inconsistent round-trip times make it difficult for the receiver to decide what latency it expects to see when the sender is not sending anything. This can deceive the receiver into believing it is being sent a 1, when in reality it is being sent a 0. Network traffic can also cause the bursts of the sender to have a different latency every time. This can lead the signal to be inconsistent from bit to bit as the sender has to wait for a different duration for each burst.

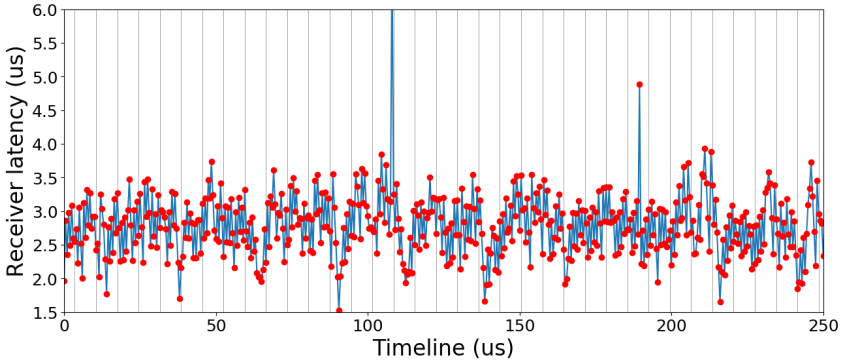
In order to model a realistic network load on the intermediary, we use the `ib_read_bw` benchmark, which is part of the Perftest [20] package for RDMA profiling. This benchmarks issues RDMA Read requests. We choose it because we can exactly specify how much network bandwidth it generates. We launch `ib_read_bw` from two machines, which are different from the three machines we use for the channel. Together, the two machines generate network bandwidth directed to the intermediary, equal to 70% of the intermediary's total link bandwidth. This load allow us to look at the performance of the channel under a relatively heavy load, without stressing the intermediary. We note earlier that stress testing is not useful for out evaluation because data center machines are often underutilized [11].

In Figure 5.2, we show the signal observed by the receiver in the presence of network load. The first thing we notice compared to the unloaded case is that the probe latencies of the receiver are higher. This is due to queueing in the intermediary's NIC as a result of the network load. While the signal is visible for burst sizes 64 and 128, it becomes impossible to distinguish bursts for burst size 32.

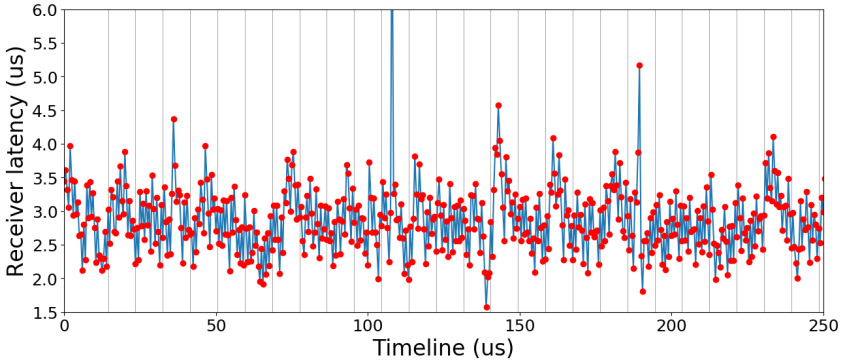
In Table 5.2 we show the throughput and the error rate of the channel for burst sizes 64 and 128. Compared to the isolated performance (Section §5.2), the throughput decreases. This is because bursts take longer due to the network load, and quiet periods have to be adjusted to match the burst period as well. The error rate also goes up significantly. This is expected since the network becomes more unpredictable.

5.4 Covertness and Tail Attack

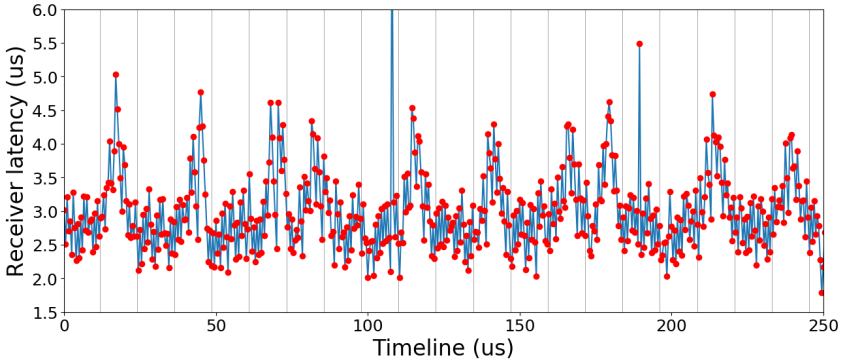
Modern processors implement hardware counters which can be used to monitor the performance of the system, such as the CPU utilization and memory bandwidth. Such



(a) Burst 32



(b) Burst 64



(c) Burst 128

Figure 5.2: Parts of the Bankrupt channel signal in our private cluster in the presence of network load.

Burst Size	1024	512	256	128	64	32	16
Bandwidth (GB/s)	1.04	0.97	0.82	0.67	0.48	0.31	0.18

Table 5.3: Memory bandwidth while the channel is running in isolation in the private cluster. The maximum bandwidth per bank is 1.25GB/s.

counters can be used by system administrators to identify any unusual behaviour or to track reasons for performance degradation. If the Bankrupt attack has a significant effect on any of these counters, then there is a risk of it being detected and halted. In order to evaluate the stealthiness of the Bankrupt channel, we explore how relevant existing counters are affected by the attack.

In this section, we use a message consisting only of ”1s”. In this way, the sender is always sending bursts. This generates considerably more load in the intermediary's memory than the message we were using to evaluate the throughput and the error of the channel before. Although this is not a very realistic message, it tells us how stealthy the attack is in the worst case.

Perf [5] is one of the most commonly used tools for profiling Linux systems. The memory of the intermediary is the main component that the Bankrupt attack exercises pressure on. On the processor found in the machines in our private cluster, we can use Perf to monitor two relevant memory metrics on the intermediary: the memory bandwidth and the cache hit/miss rate.

In Table 5.3, we show the memory bandwidth measured by Perf, while the channel is running in isolation. In modern systems with many banks where the bandwidth per channel can be around 20GB/s, the attack has a negligible impact on the memory bandwidth counter measured by Perf. The cache hit/miss rate is not affected by the attack because RDMA requests do not get cached.

On the other hand, since the maximum bank bandwidth for our system is around 1.25GB/s, the Bankrupt channel utilizes 24% to 53% of the total bank bandwidth for burst sizes 32 to 128, respectively. This means that the attack can be noticed by counters which look at the memory bandwidth per bank. The Bankrupt attack causes the bank used by it to have a much higher utilization than any of the other banks. This is unusual as banks are designed to have close to equal utilization. However, as far as we know, such per-memory-bank counters do not exist in Perf.

In order to further evaluate the covertness of the attack, we consider how the attack influences the memory access time of the intermediary machine. If the impact of the Bankrupt attack on the memory access time is high, then other applications will start to experience a performance drop, which might reveal the attack. We implement a benchmark that performs random memory accesses in a 2GB memory region and records the latencies that it sees.

In Figure 5.3, we show the median and the tail memory latency on our cluster while the channel is not running and while the channel is running with different burst sizes. We see that increasing the burst size from 4 to 2048 does not influence the memory latency up until the 99th percentile. For burst size 32, we observe that the 99.9th and

the 99.99th percentiles increase by approximately 20% and 70%, respectively. The Bankrupt attack influences the very heavy tail of the memory access time, which is likely not enough to significantly affect other applications and raise an alert to the administrators.

As far as we are aware, there do not exist counters which measure the average memory access time for each bank. Because the Bankrupt channel only affects the memory latency of one bank, such counters are likely to show a major difference among the response times of the different banks, potentially revealing the attack.

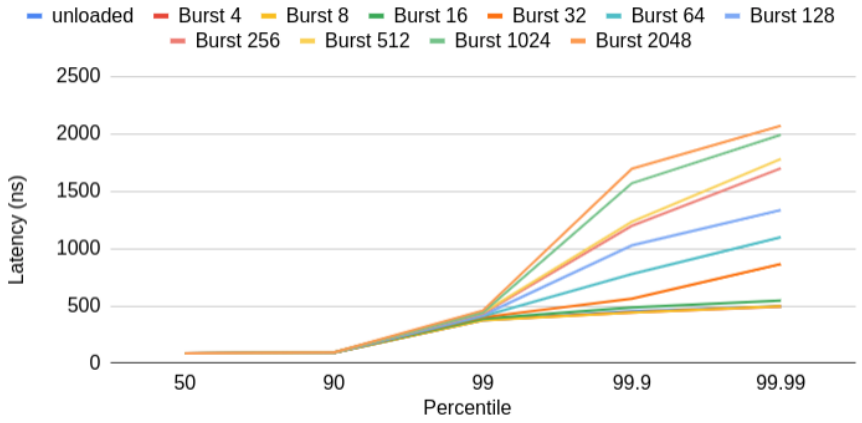


Figure 5.3: Memory access times recorded while the Bankrupt channel is running with different burst sizes in the private cluster.

5.5 Tail Attack

The tail latency of service is the small fraction of latencies which take the longest. Depending on the service and the requirements, we can define the tail latency as the 90th, 99th or even the 99.9th percentile of the observed latencies.

For interactive applications such as online shopping or searching websites, both low average latencies and low tail latencies are crucial to maximize profit. High tail latencies can violate the QoS requirement promised by the service provider and drive customers away. Given the large number of people that use such services, this can result in a massive financial loss. For example, Amazon reports that 100 milliseconds of extra load time cost them 1% in sales [28].

There can be various non-malicious causes of tail latency in large-scale services which run in the cloud. Clouds constitute of machines with different hardware and a slow component in one machine can cause high tail latency for the user. Moreover, hot items which are accessed by many users simultaneously can generate large queues in the machines which store these items. Last but not least, component failure are frequent in large cloud infrastructures, which might cause a delay in some requests. Cloud providers have invested in identifying when high tail latency occurs and the reasons for it, as well as how to minimize it [23, 13, 37].

Tail attacks are a form of denial of service (DoS) attacks, which often target web applications which run on distributed cloud infrastructures [27]. Their goal is to make the attacked service unusable or frustrating to use by customers by making its tail latency exceptionally high. Conventional DoS attacks use a brute force approach of sending more requests than the service can handle, which makes them obvious. On the other hand, tail attacks are more subtle because they only need to send enough of the right requests to affect 1% of the requests or less. This makes detecting them and eliminating them harder.

The Bankrupt covert channel causes one of the banks in the intermediary machine to have high latency when the sender has issued a burst. This means that the sender can cause the bank to have a high latency at all times by continuously issuing bursts. In turn, assuming that a service running on the intermediary is using all banks equally[1], then only a fraction of the requests will be delayed. These are the requests that go to the bank that the sender is flooding.

In Figure 5.3 we observe that for big burst sizes such as 1024 and 2048, the 99.9th and the 99.99th percentiles of the memory latency become significantly larger than in the unloaded case. To quantify, 0.1% of memory accesses become 3 times slower and 0.01% of memory accesses become 4 times slower. This makes the Bankrupt attack usable in a commercial cloud setting as well, where tenants share the cloud hardware. A malicious tenant can use the Bankrupt channel to mount a tail attack against a competing application and make profit off of that.

5.6 CloudLab Results

In this section, we show the experiments we carry out on the CloudLab Apt cluster. We present these results in a separate section, because they show the behaviour of the channel in a realistic data center environment.

The utilization of the nodes on the cluster is the only metric which gives us an idea of the load present in the cluster. At the time of testing, the utilization was 97%. Still, is not always a telling metric of the load present in the cluster, because some tenants might be occupying the machines without executing any tasks on them. It is also possible for other tenants to be running local processes on nodes, which our attack does not use, which does not influence the behaviour of our channel.

Although the environment in which we run the experiments is definitely not isolated, it is impossible to precisely gauge the level of load present in the cluster. For this reason, the results we obtain cannot reliably be compared to any *single* results from our cluster, be it the isolated results (Section §5.2), or the results in a loaded environment (Section §5.3).

Figure 5.4 shows parts of the signal for different burst sizes in CloudLab. The first thing that we notice is that round-trip times in CloudLab are higher than the ones in our private cluster. Moreover, bursts cause a higher latency in the memory. This can be

[1] This is the whole point of having banks in the first place.

Burst Size	Period (μs)	Throughput (Kbps)	Error rate (%)
128	22	45.45	0.5%
64	13.06	76.57	0%
32	8.53	115.61	0%
16	6.35	157.48	0%
8	4.89	204.5	3.5%
4	4.35	229.89	47%

Table 5.4: Throughput and error rate of the Bankrupt channel for different burst size in the CloudLab cluster. The period is the round-trip time per burst, and respectively, the time the sender remains quiet.

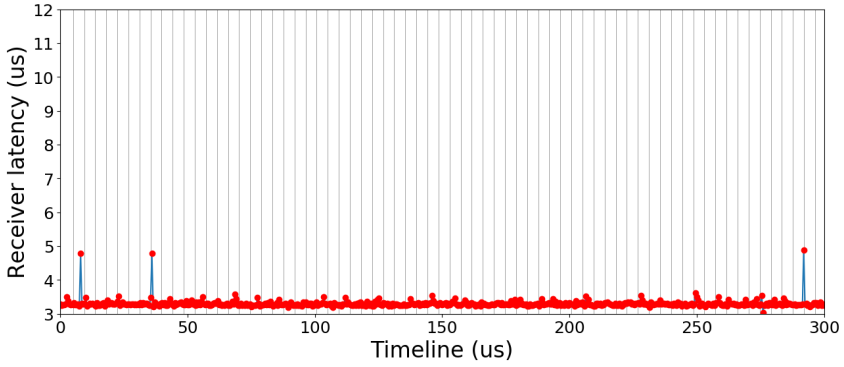
cause by older memory that CloudLab machines use (DDR3) or network load present in the cluster.

However, what is more interesting to see is that the bursts have a much more regular shape in CloudLab and all of them are very similar to each other. They almost look like perfect triangles. This makes bursts much easier to distinguish and greatly helps the decoder in identifying them.

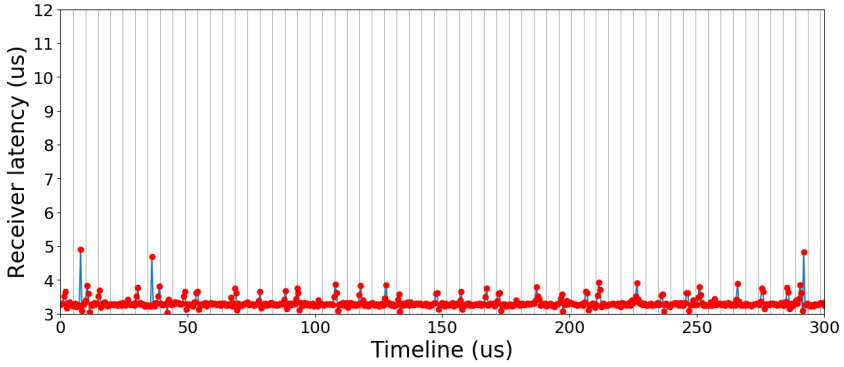
In Table 5.4, we can see the throughput and the error rate for different burst sizes. Because of the larger round-trip times in CloudLab and the larger latencies that burst incur in the intermediary compared to our private cluster, we can reduce the bursts size to 8, while only incurring an error of 3.5%. We see that the error rate is a lot better than in our private cluster results. This is made possible by the predictable shape of the bursts in CloudLab, which allows the decoder to robustly identify them.

Nevertheless, when we set the burst size to 4, we reach a limit, where bursts hardly cause the latency observed by the receiver to increase. In this way, almost all bits get predicted as 0, which is the reason for the 47% error rate. Even though the error rate is 47% for burst size 4, the channel is not functional for this burst size, because almost all bits are predicted as 0's. This means that for a message consisting only of 1's, the channel will have close to a 100% error rate.

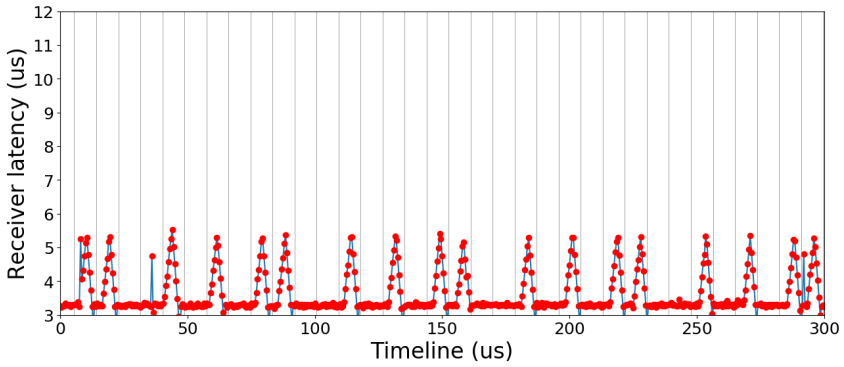
Finally, the best result is when the burst size is 8. We achieve a throughput of 204.5Kbps with an error rate of 3.5%.



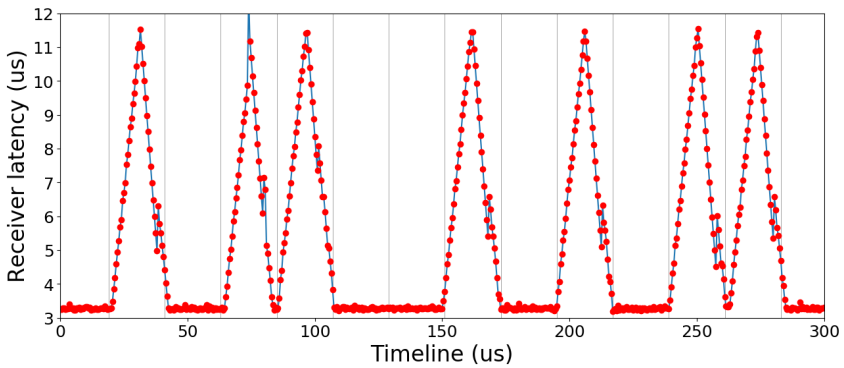
(a) Burst 4



(b) Burst 8



(c) Burst 32



(d) Burst 128

Figure 5.4: Parts of the Bankrupt channel signal on the CloudLab cluster.

Chapter 6

Future Work

In this chapter, we give pointers for future work on the channel and summarize and interpret the results we obtain.

6.1 Future Work

We report the performance of the channel in terms of throughput and error rate. For a fully working mechanism, it is necessary to provide a method for error checking or error correction. For example, the sender and the receiver can use a checksum. When the receiver finds that the payload has been corrupted, it needs a way to notify the sender to retransmit the message. In this way, we will be able to evaluate the "goodput"[1] of the channel.

Error checking and correction is part of a larger task is to develop a full protocol between the sender and the receiver. We partially do this by including a preamble in the sender's message. However we can design a protocol, which fully specifies the structure, the length and the contents of each message. Although such a protocol is not the focus of this work, which aims to demonstrate the fundamental functionality of the channel and evaluate its behaviour under different cluster conditions, it will carry the channel to the next level where structured data can be transmitted over it.

In the future, we would also like to further optimize the channel. For example, the memory requirement of the channel can be reduced by making sure that the addresses that the sender and the receiver send RDMA Reads to are all located on different rows. This will result in row conflicts in the intermediary's bank, which have higher latency than average memory requests (as these are not all row conflicts). In this way, smaller bursts can be used by the sender to achieve the same latency in the intermediary's bank.

We can improve the stealthiness of the channel by inverting the signal. Inverting means that the sender will send bursts to send a 0, and keep quiet to send a 1. We note that only bursts cause traffic in the intermediary's bank. The sender can thus inspect the

[1]The "goodput" is the throughput of useful information in the channel. It takes into consideration the overhead of headers, checksums, and retransmissions due to errors.

message that it is sending and see if 1s or 0s are more common. Then, it will use a burst to send the less common bit. This results in less bandwidth generated by the channel and makes the attack more stealthy. In this case, the sender and the receiver need to communicate somehow if the signal is inverted or not. This can be part of the protocol that we discussed.

The bandwidth of the RDMA network can be 10 times larger than the bandwidth of a single bank. This means that the channel can scale to using multiple banks in the same intermediary with the current technology. In this way, the sender and the receiver can use multiple threads to increase the bandwidth of the channel linearly with the number of banks used.

Last but not least, it might be possible to enhance the throughput of the channel by shortening the period for which the sender remains quiet to transmit a 0. Strictly speaking, it is not required for the period for a 0 and a 1 to be the same, even though changing it will definitely complicate the decoding process. Although the period for a 1 is determined by the round-trip of the bursts, we can arbitrarily set the period for a 0. It would not make sense to make the 0 period larger, as this would reduce the throughput of the channel, but we can make it smaller. In this case, we have to be careful because setting the waiting period to be too small can make 0's indistinguishable in the receiver. In the decoding stage, the receiver needs to know the period for a 0. If it does not "look like" the current chunk that it is decoding is a 1, then the receiver can use a shorter period to decide if the chunk is a 0. This decoding procedure can result in one error propagating to the following bits in the payload and it can cause more harm than good. Either way, it is definitely worth exploring.

Chapter 7

Conclusion

Governmental agencies and many companies have their own private clouds, which they use to store confidential government data or sensitive customer information. Leaking such data can cause major problems with GDPR or international conflicts. This is why cloud owners invest heavily to make sure their data is protected from illegitimate accesses.

Covert channel attacks pose a serious threat to the security of data as they enable the exchange of information between entities which should not be allowed to communicate. Our work introduces the Bankrupt covert channel. The Bankrupt attack establishes a covert channel in RDMA-connected clusters. The sender and the receiver are located on different physical machines. The attack creates a timing channel in one of the memory banks of a third machine in the cluster.

As part of our work on the channel, we present a procedure to identify a set of virtual addresses that map to the same bank in a system's memory. We describe how to tune the channel to achieve a robust signal and explain how the signal can be decoded. We also discuss potential mitigation techniques that can be implemented to prevent the attack.

We show that the attack can be launched without any modifications on our private cluster and the CloudLab cluster. This entails that the Bankrupt attack can generalize to various RDMA-enabled infrastructures, because it exploits fundamental design features of main memory and RDMA which are present in virtually all modern systems.

Our attack achieves comparatively high throughput. Even though 204.5Kbps may not sound like a lot, we must consider that sensitive information such as authentication details, private keys or credit card details are usually items that are not bigger than a few KB. For most covert channel, being able to transfer such information in several seconds is considered a success. The Bankrupt channel can do this in less than a second. Nevertheless we have to account for the fact that implementing error correction or error checking will cause overhead, reducing the effective bandwidth of the channel.

We show that local load on the intermediary's memory does not affect the performance of the Bankrupt channel. This makes the attack effective against machines in the cloud

which run mostly local applications.

Network loads manifests in a greater problem, where small burst sizes become indistinguishable. As a result, we have to use larger burst sizes. Thus, the throughput diminishes by 30% and the error rate becomes very significant – close to 15%. This decrease in performance will become even prominent when error correction is implemented, because the overhead of correction codes will be considerable given the large error.

Having to use larger bursts in the presence of network load also soars the bandwidth consumed by the channel. This makes the channel easier to detect if we consider it alone, but the network load actually enables the channel bandwidth to blend in with the external load.

We show that the channel is undetectable by existing Perf counters which measure aggregate statistics such as memory bandwidth and average memory access time. However, the channel can be detected by counters which measure per-bank statistics. Bank statistics are difficult to monitor without hardware assistance. This is because Intel processors do not reveal the mapping from memory addresses to banks. At the same time, it is unlikely that Intel will provide such counters as they can easily be used to reverse engineer the bank mapping.

Because of the high throughput, covertness, and resistance to noise, the Bankrupt attack poses a security threat to cloud owners. Public clouds providers are vulnerable as well, because the tail attack can be used to violate the QoS guarantees that these providers promise to their customers. As far as private clouds are concerned, it can admittedly be difficult to compromise a machine in the Pentagon in order to obtain some information in the first place. But if we could do that, we could use Bankrupt to exfiltrate this data to a machine connected to the public Internet.

Covert channels in general have become common due to the timing differences present in many contemporary hardware components – caches, memory, memory bus, routers. Like we discuss in the mitigations for the Bankrupt channel, these timing differences are usually tied with performance and they are hard to remove without degrading the performance. Because of this, hardware engineers, computer engineers, and even software engineers need to work hard to minimize the possibility for an attack.

It is likely that more covert and side channels will be discovered in the near future. In our work with the Bankrupt attack, we find that keeping design decisions such as the memory controller's hashing function physical addresses to banks does not significantly help against attackers. Such design decisions can often be reverse engineered. What is more, keeping designs proprietary hinders their security properties from being evaluated by the wider community, which can disclose potential vulnerabilities early and suggest how to fix them.

Bibliography

[1] CloudLab. Available at `www.cloudlab.us`.

[2] Google Cloud Hardware. Available at `https://cloud.google.com/compute/docs/cpu-platforms`.

[3] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE symposium on security and privacy (SP)*, pages 987–1004. IEEE, 2016.

[4] James Charles, Preet Jassi, Narayan S Ananth, Abbas Sadat, and Alexandra Fedorova. Evaluation of the intel® core™ i7 turbo boost feature. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 188–197. IEEE, 2009.

[5] Arnaldo Carvalho De Melo. The new linux'perf'tools. In *Slides from Linux Kongress*, volume 18, pages 1–42, 2010.

[6] Intel Data Direct. I/o technology (intel ddio).

[7] Aleksandar Dragojevic, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast remote memory. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 401–414, 2014.

[8] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299. Springer, 2016.

[9] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[10] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! a fast, cross-vm attack on aes. In *International Workshop on Recent Advances in Intrusion Detection*, pages 299–319. Springer, 2014.

[11] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. *SIGARCH Comput. Archit. News*, 43(3S):158–169, June 2015.

[12] Michael Kurth, Ben Gras, Dennis Andriesse, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Netcat: Practical cache attacks from the network, 2020.

[13] Jialin Li, Naveen Kr Sharma, Dan RK Ports, and Steven D Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, 2014.

[14] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE, 2015.

[15] Margaret Rouse. How NVMe-oF advantages are transforming storage, 2017. Available at `https://searchstorage.techtarget.com/definition/Remote-Direct-Memory-Access`.

[16] Deborah T Marr, Frank Binns, David L Hill, Glenn Hinton, David A Koufaty, J Alan Miller, and Michael Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1), 2002.

[17] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse engineering intel last-level cache complex addressing using performance counters. In *International Symposium on Recent Advances in Intrusion Detection*, pages 48–65. Springer, 2015.

[18] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. C5: Cross-cores cache covert channel. In *DIMVA*, 2015.

[19] MEG ANDERSON. Pentagon Awards $10 Billion Contract To Microsoft Over Front-Runner Amazon, 2019. Available at `https://www.npr.org/2019/10/26/773706177/pentagon-awards-10-billion-contract-to-microsoft-over-front-runner-amazon`.

[20] Mellanox. Mellanox Perftest Package, 2017. Available at `https://community.mellanox.com/s/article/perftest-package`.

[21] Micron Technology Inc. DDR4 SDRAM datasheets, 2018. Available at `www.micron.com/products/dram/ddr4-sdram`.

[22] Microsoft Corporation. Introducing the new HBv2 Azure Virtual Machines for high-performance computing, 2019. Available at `https://azure.microsoft.com/en-us/blog/introducing-the-new-hbv2-azure-virtual-machines-for-high-performance-computing/`.

[23] Pulkit A Misra, María F Borge, Íñigo Goiri, Alvin R Lebeck, Willy Zwaenepoel, and Ricardo Bianchini. Managing tail latency in datacenter-scale file systems under production constraints. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–15, 2019.

[24] Colin Percival. Cache missing for fun and profit, 2005.

[25] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. {DRAMA}: Exploiting {DRAM} addressing for cross-cpu attacks. In

25th {USENIX} Security Symposium ({USENIX} Security 16), pages 565–581, 2016.

[26] Robert Ricci, Eric Eide, and CloudLab Team. Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications. *; login:: the magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[27] Huasong Shan, Qingyang Wang, and Calton Pu. Tail attacks on web applications. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1725–1739, 2017.

[28] Steve Olenski Rouse. Why Brands Are Fighting Over Milliseconds, 2016. Available at `https://www.forbes.com/sites/steveolenski/2016/11/10/why-brands-are-fighting-over-milliseconds/#1320be5a4ad3`.

[29] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory deduplication as a threat to the guest os. In *Proceedings of the Fourth European Workshop on System Security*, pages 1–6, 2011.

[30] Shin-Yeh Tsai, Mathias Payer, and Yiying Zhang. Pythia: Remote oracles for the masses. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 693–710, Santa Clara, CA, August 2019. USENIX Association.

[31] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22nd annual international symposium on Computer architecture*, pages 392–403, 1995.

[32] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud. *IEEE/ACM Transactions on Networking*, 23(2):603–615, 2014.

[33] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. Security implications of memory deduplication in a virtualized environment. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.

[34] Xing Pan, Frank Mueller. The Colored Refresh Server for DRAM, 2019. Available at `https://isorc2019.github.io/html/slides/Session2_Mueller.pdf`.

[35] Yuval Yarom and Katrina Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 719–732, 2014.

[36] Yuval Yarom, Qian Ge, Fangfei Liu, Ruby B Lee, and Gernot Heiser. Mapping the intel last-level cache. *IACR Cryptology ePrint Archive*, 2015:905, 2015.

[37] Timothy Zhu, Alexey Tumanov, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. Prioritymeister: Tail latency qos for shared networked storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, 2014.