Hypercall Survey

To perform load testing, it is necessary to be able to execute hypercalls correctly, unlike in Section 6.3, where non-existing hypercalls were issued. Unfortunately, as Section 7.1 described, it was not possible to retrieve trustable information about which hypercalls happen during normal operation, which would have been useful to identify suitable calls. This section provides a survey of the hypercalls documented in Hyper-V's functional specification [14].

Each call is first described and then judged depending on how well it fits the requirements of the hypercall testing framework and the load testing workload. The framework's restrictions rule out rep calls and calls with variably-sized headers. To be able to be used effectively in stress testing, a hypercall has to have either no parameters or the values to be passed have to be known, such that the hypercall does perform the work it is supposed to do. Additionally, the call has to be able to be executed repeatedly, without causing any (intentional) malfunction to the hypervisor.

- **[SUITABLE] HvExtCallQueryCapabilities**: This is one of the extended hypercalls supported by Hyper-V. It does not take any parameter. All it does is output a value called `Capabilities`, which is 8 bytes large. The first bit indicates whether the extended hypercall "HvExtCallGetBootZeroedMemory" is available to call. The other 63 bits are reserved. This hypercall is suitable for testing, but it probably is very inexpensive for the hypervisor to processes.
- **[UNSUITABLE] HvSetVpRegisters**: This call takes a list of register name-value pairs, and sets those values in the registers of the specified guest partition. During normal execution, this might produce errors or crashes in the guest partition. However, the call is unsuited in the first place because it is a rep call.
- **[UNSUITABLE] HvGetVpRegisters**: This is a pendant to the `HvSetVpRegisters` call. It takes a list of register names, looks up their values in the specified guest partition, and outputs those to the caller. This should be safe to execute because nothing is actively changed, and interesting because of the register-checking workload, however this unsuitable due to being rep call, as well.
- **[UNSUITABLE] HvStartVirtualProcessor**: When a guest partition with multiple cores is booting, it has to start the other processor cores. This hypercall provides a paravirtualized way of doing this. The ID of the partition (-1 can be used to refer to own partition) and the index of the processor have to be passed, along with values for each of the registers. When this call is executed during normal operation, it is probably either going to fail to execute or crash the operating system.
- **[UNSUITABLE] HvGetVpIndexFromApicId**: Usually, virtual processors can detect their virtual processor index by reading an MSR. However, this is not possible when the processor is not yet started. This hypercall allows retrieving the virtual processor index using its ID of the Advanced Programmable Interrupt Controller (APIC). Unfortunately, it is a rep call, taking a whole list of APIC IDs, and thus unsuitable.
- **[UNSUITABLE] HvSwitchVirtualAddressSpace**: This hypercall is the paravirtualized method of writing to the CR3 register: it changes the page table used for memory translation. However, unlike a CR3 write, the TLB is not flushed implicitly. Changing the address space is likely to quickly cause crashes due to illegal or wrong memory accesses, rendering this call unsuitable.
- **[SUITABLE] HvFlushVirtualAddressSpace**: This call provides a paravirtualized way to flush TLBs. So, together with the `HvSwitchVirtualAddressSpace`, it can be

used to perform the equivalent of a CR3 write. However, this call cannot only flush the TLB of the processor core executing the call but also of other cores of the same partition. Usually, a page table address, and a bitmask describing the processor to be flushed, have to be specified, which are not known before runtime. It is possible to specify the flag `HV_FLUSH_ALL_PROCESSORS` to flush the TLBs of all virtual processors. Similarly, `HV_FLUSH_ALL_VIRTUAL_ADDRESS_SPACES` can be used to clear all TLB entries. Combining both flags results in the call flushing all values from all TLBs in the partition. This call is suitable to be used in testing campaigns, and interesting because of its TLB-flushing workload.

- **[UNSUITABLE] HvFlushVirtualAddressSpaceEx**: This is a rep call version of the call `HvFlushVirtualAddressSpace`, and allows to define a processor set whose TLBs should be flushed. Due to its rep call nature, and the existence of the non-rep version, this call is unsuitable.
- **[UNSUITABLE] HvFlushVirtualAddressList**: This is yet another version of the `HvFlushVirtualAddressSpace` call, with the possibility to specify virtual address ranges. All TLB entries overlapping with those ranges are flushed. As with the previous call, due to its rep call nature, and the existence of the non-rep version, this call is unsuitable.
- **[UNSUITABLE] HvFlushVirtualAddressListEx**: This is a further variant of `HvFlushAddressSpaceList`, but takes a set of virtual processor IDs. It is a rep call, and thus, unsuitable.
- **[MAYBE SUITABLE] HvTranslateVirtualAddress**: This call can be used by the root partition to translate a GVA of a guest partition to a GPA. It is unknown which address the page table will be mapped when the call is executed by a campaign, but even for unmapped addresses, the hypervisor will have to perform a lookup in the page table, so, this call might be suitable.
- **[SUITABLE] HvExtCallGetBootZeroedMemory**: This call allows a partition to find out which memory pages have been zeroed by the hypervisor to avoid zeroing them twice. This call is suitable to be executed, but will probably not be expensive to process.
- **[UNSUITABLE] HvAssertVirtualInterrupt**: This call allows the root partition to trigger an interrupt in a guest partition. Due to unknown values needed for the guest partition ID, this call is classified as unsuitable.
- **[MAYBE SUITABLE] HvSendSyntheticClusterIpi**: This call is also concerned with interrupts. It sends interrupts with a given interrupt vector to the processors defined by a bitmask. The interrupt vector has to be in the range between 0x10 and 0xff, which could be tried to be executed. Thus, this call is classified as being maybe suitable.
- **[UNSUITABLE] HvSendSyntheticClusterIpiEx**: The same as the previous call `HvSendSyntheticClusterIpi`, but with a variably-sized processor set as input. It is a rep call and thus unsuitable.
- **[UNSUITABLE] HvRetargetDeviceInterrupt**: The documentation states that this call retargets a device interrupt, presumably to another virtual processor. Due to the lack of understanding of the purpose of the call, as well as its parameters, it is unsuitable.
- **[MAYBE SUITABLE] HvPostMessage**: This hypercall posts a message of up to 240 bytes to a connection specified by their ID. The hypercall monitor, if working, could

be used to identify the IDs of open connections. However, it is unknown what will happen when random messages are delivered to them. Subsequently, this call is rated to be maybe suitable.

- [**MAYBE SUITABLE**] **HvSignalEvent**: Similarly to `HvPostMessage`, this operates on a connection ID. No message is specified here, only a flag, which is transported to the other end of the connection, which is then also notified using a synthetic interrupt. Again, it could be tested how the system reacts to random event signaling; thus, the call is maybe suitable.
- [**SUITABLE**] **HvNotifyLongSpinWait**: If a partition has waited a long time to acquire a spinlock, which another processor core in the same partition holds, it can issue this call to hint the hypervisor that it might adjust its scheduling. Input is a number describing how long the processor has been waiting for the lock. This call is suitable and might be interesting if the hypervisor actually does take action and checks its scheduling.
- [**UNSUITABLE**] **HvModifyVtlProtectionMask**: It is unclear what this call actually does, and with it being a rep call, it is unsuitable anyways.
- [**UNSUITABLE**] **HvEnablePartitionVtl**: As the previous call, this one is related to Virtual Trust Level (VTL) management, which is an unclear concept at the point of writing. Subsequently, this called is marked as unsuitable.
- [**UNSUITABLE**] **HvEnableVpVtl**: The same goes for this hypercall; it is unsuitable due to a lack of understanding.
- [**MAYBE SUITABLE**] **HvVtlCall**: This call is specified to change from one VTL to the next higher. It is unclear what the result of execution will be. As there are neither parameters nor output values, though, it might be tried.
- [**MAYBE SUITABLE**] **HvVtlReturn**: The same goes as for `HvVtlCall`, but this call reduces the VTL. No parameters, so it might be suitable.
- [**UNSUITABLE**] **HvFlushGuestPhysicalAddressSpace**: This call is supposed to be used with nested virtualization. It flushes the TLBs that cache translations from second-layer GPAs to GPAs. An address space ID has to be passed, of which none are known. Thus, the call is unsuitable.
- [**UNSUITABLE**] **HvFlushGuestPhysicalAddressList**: Similar to the hypercall `HvFlushGuestPhysicalAddress`, but additionally, a range of GPA pages can be specified. It is unsuitable for the fact that it requires an address space ID, and additionally, for being a rep call.

To provide a better overview of which calls are suitable, maybe suitable, or unsuitable to be used for testing with the current framework implementation and knowledge about Hyper-V, the categorizations are presented in Table 7.1.

This section has provided the details necessary achieve the first part of the fifth goal:

G5: *Identify hypercalls suited for repeated execution and evaluate the impact of different load levels.*

The calls **HvExtCallQueryCapabilities**, **HvFlushVirtualAddressSpace**, as well as **HvExtCallGetBootZeroedMemory**, and **HvNotifyLongSpinWait** have been found suitable to be used for load testing. Six other documented hypercalls have been judged to be potentially suited as well; however, further evaluation of their behavior will be necessary to clarify.

Hypercall	√	?	X
HvExtCallQueryCapabilities	X		
HvSetVpRegisters			X
HvGetVpRegisters			X
HvStartVirtualProcessor			X
HvGetVpIndexFromApicId			X
HvSwitchVirtualAddressSpace			X
HvFlushVirtualAddressSpace	X		
HvFlushVirtualAddressSpaceEx			X
HvFlushVirtualAddressSpaceList			X
HvFlushVirtualAddressSpaceListEx			X
HvTranslateVirtualAddress		X	
HvExtCallGetBootZeroedMemory	X		
HvAssertVirtualInterrupt			X
HvSendSyntheticClusterIpi		X	
HvSendSyntheticClusterIpiEx			X
HvRetargetDeviceInterrupt			X
HvPostMessage		X	
HvSignalEvent		X	
HvNotifyLongSpinWait	X		
HvModifyVtlProtectionMask			X
HvEnablePartitionVtl			X
HvEnableVpVtl			X
HvVtlCall		X	
HvVtlReturn		X	
HvFlushGuestPhysicalAddressSpace			X
HvFlushGuestPhysicalAddressSpaceList			X

Table 7.1.: Suitablility of Documented Hypercalls for Testing