Computer Communications 85 (2016) 14–27

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

DevCom: Device communities for user-friendly and trustworthy communication, sharing, and collaboration

H.V. Hansen[a,*], V. Goebel[a], T. Plagemann[a]

University of Oslo, Department of Informatics, Gaustadalléen 23 B, N-0373 Oslo, Norway

ARTICLE INFO

Article history:
Received 10 December 2014
Revised 7 January 2016
Accepted 13 February 2016
Available online 23 February 2016

Keywords:
Mobile applications
Peer-to-peer
Trustworthy communication
Virtual Private Networks

ABSTRACT

We present the design, implementation and evaluation of DevCom, a network system that provides users with a trustworthy and user-friendly way to communicate, share and collaborate among distinct groups of devices simultaneously, e.g., home devices, work devices and friends' devices. DevCom is trustworthy, because reliability, security and privacy issues are automatically taken care of with state-of-the-art cryptography. User-friendliness is ensured through self-configuration, persistent connections in advent of mobility, and by automatically solving problems that network address translation (NAT) and firewalls introduce. An experimental evaluation shows that (1) off-the-shelf applications such as games perform in the same way as they do without DevCom, (2) latency, throughput and processing overhead is low and unnoticeable to users, and (3) persistent connections are automatically supported on devices with changing addresses. A critical analysis from three user perspectives, i.e., a novice, an intermediate user, and an application developer, highlight the user-friendliness of DevCom for a broad range of users.

1. Introduction

People are surrounded by computing devices with ubiquitous access to data networks such as Wi-Fi and cellular networks. This is making Mark Weiser's vision of ubiquitous computing [1] a reality in the quantitative aspect. It is common to own smart-phones, laptops, desktop computers, tablets, and specialized computers such as media centers, and use additional devices at school, work and when visiting friends and family. However, the qualitative part of Weiser's vision, regarding unconscious human-computer interaction based on seamless communication, sharing, and collaboration between devices, is still unfulfilled. Files (e.g., music libraries and text documents), peripherals (e.g., printers and web-cams) and services (e.g., remote desktop and secure shell) are not always accessible on the devices currently available to users. Access to such remote resources is even more cumbersome if proper security and privacy protection mechanisms are in place.

In order to make life less complicated for users, this work presents a network system called DevCom that assists users to easily organize devices in multiple trustworthy communities. Communities are small groups of devices (later called device communities) with common trust policies, where ubiquitous communication, sharing, and collaboration among the devices are assured. A device community is providing security and privacy protection and can be seen as a generic trustworthy platform for all types of network applications. Devices can be part of many device communities at the same time and take advantage of the different services and applications in each device community. A mobile device on a public Wi-Fi can, e.g., securely access a shared spreadsheet document at work and print it using the family printer at home. To minimize user interaction, DevCom even works seamless when mobile devices change their IP address, and in the presence of firewalls and network address translation (NAT).

One question to investigate is whether this type of device management requires new research or if sufficient solutions already exist. The answer is that some of the DevCom features can be achieved by combining existing technologies, but at the expense of user-friendliness. It is possible to configure port forwarding and static IP addresses for devices behind NAT, dynamic name server updates for devices using Dynamic Host Configuration Protocol (DHCP), multiple Virtual Private Network (VPN) tunnels, and mobile IP, but "only the most dedicated, desperate, or geeky will go to this trouble" [2]. Simpler techniques typically rely on third-party services, such as Hamachi [3]. These services can cost money, introduce downtime when the service providers experience failures, be slow if the service providers are located across the globe, and perhaps most importantly, raise serious privacy concerns.

Trustworthy and user-friendly is often considered to be a contradiction, because security features normally incorporate constraints and obligations. One of the core goals of this work is to

* Corresponding author. Tel.: +47 22852876; fax: +47 22852401.
E-mail addresses: hansvh@ifi.uio.no (H.V. Hansen), goebel@ifi.uio.no (V. Goebel), plageman@ifi.uio.no (T. Plagemann).

http://dx.doi.org/10.1016/j.comcom.2016.02.001
0140-3664/

achieve trustworthiness while maintaining user-friendliness. DevCom relies on self-configuration in order to minimize user interaction and reduce the risk of misconfiguration. Users might choose poor passwords, because they are easy to remember, and use potentially fatal configuration, e.g., if forced to define IP ranges, choose dedicated super peers or manage cryptographic key distribution. By taking the configuration burden away from the user, the security and user experience is improved at the same time. The only task to be performed by users is to name their device communities and let devices join and leave when desired. The rest of the difficult configuration and maintenance is performed by DevCom. A critical part of the self-configuration mechanism is a novel IP address assignment algorithm using cryptographic keys to yield permanent and unique addresses for each device.

In many ways a device community is similar to a VPN. Both imply a group of distributed devices in a trustworthy virtual network. Member admission is controlled, communication is encrypted and data is only transmitted to whom it concerns, ensuring privacy for the users. However, there are also substantial differences between device communities and VPNs. First, a device community is completely decentralized and independent of third party servers. Second, and arguably more important, VPNs do not allow a device to be part of several networks at the same time. We regard coexisting networks as essential in ubiquitous computing. Using applications and services in different networks at the same time is required in scenarios like the one described above where a user's current task spans networks, i.e., working from a public network on a document in a private home network and printing it in another completely separate work network. It is also beneficial when two or more tasks are performed simultaneously in different networks, e.g., if a user is streaming audio from a friend while troubleshooting a family member's computer using remote desktop. Furthermore, not all devices a user has access to are equal in terms of trust or demand for files, peripherals and services. For example, devices used for work should not necessarily have access to the same services as home devices, and friends' devices are not trusted in the same way that family devices are. Similarly, mobile devices should not necessarily have the same privileges as stationary devices, and workstations can have different needs than servers. With existing network technology, if a user enables a service on a device, e.g., a web server listening on port 80, all incoming requests to that service are treated equally, with no simple means to allow some devices to connect, while denying others. If a user of DevCom wants to deny access to a service (such as a web server) in one device community while allowing it in another, this is easily done using off-the-shelf firewalls with graphical user interfaces where the user can choose the application to restrict from a menu. This is predominantly relevant if the application or service does not have its own authorization features, i.e., unlike ssh.

The first contribution of this work is the device community concept and approach for organizing computing devices into multiple, concurrent groups. Device communities facilitate differentiated communication, sharing, and collaboration among computing devices, supporting existing network applications and services including games, audio/video streaming, file and printer sharing, remote desktop, ssh, etc. Few existing systems have the ability to allow devices to take part simultaneously in independent secure networks. The second contribution is a novel self-configuration mechanism. The self-configuration alleviates users from burdening and potentially unsafe configuration choices. Users are the most competent when it comes to grouping their devices in terms of trust, but should not be concerned with technical details, such as defining IP ranges, choosing dedicated super peers and managing cryptographic keys. The self-configuration mechanism is also used to separate the location and identification of devices, allowing seamless connection handover in mobile scenarios. The third contribution is the design of one complete, decentralized, self-organizing, user-friendly and trustworthy system without developing new security and privacy solutions. Instead, DevCom uses existing cryptography and NAT penetration solutions according to best practices. The fourth and final contribution is a prototype implementation and evaluation of DevCom. A combination of experiments, ranging from micro-benchmarks to preliminary user studies is performed in order to analyze the performance and user-friendliness of DevCom.

The remainder of this paper is organized as follows: Section 2 examines the configuration efforts necessary to achieve DevCom communication features in current networks, Section 3 reviews the related work, Section 4 describes the DevCom design, Section 5 presents the evaluation specific details and results, and Section 6 concludes the paper.

2. Configuration efforts in current networks

The configuration efforts necessary to achieve DevCom communication features in current networks are not negligible. Users of networked devices face a series of challenges when they want to use applications or services in different private networks simultaneously. The following are some needs users typically have, and the configuration efforts required to achieve those needs.

Connecting to devices without knowing their physical location or network attachment requires automatically updated IP addresses or hostnames. This can be achieved by expert users through third party services, such dynamic DNS, but only on certain devices and operating systems. Dynamic DNS typically involves signing up for an account with, e.g., noip.com, installing and configuring an update agent in all devices, and ensuring that devices are always addressed by hostname and that no local caching occurs.

Connecting to devices behind NAT, typically used in homes and workplaces, requires port forwarding or publicly exposed networks. Port forwarding is difficult if more than one device behind NAT use the same port, and exposed networks can potentially be a security threat. Furthermore, it requires that the devices behind NAT use static IP addresses. Port forwarding is configured in home and office routers' administration interfaces by choosing the port to open externally and the internal IP address to forward traffic to. The configuration has to be done for each port or port range to forward.

Roaming with seamless streaming, remote desktop and other application sessions requires mobile IP and fast handover or equivalent solutions. Configuring this is not trivial for regular users, illustrated by e.g. Cisco's 10 page configuration documentation just for setting up home and foreign agents.

Protecting one's privacy on public networks requires encryption, typically through VPN. VPN services are not necessarily difficult to configure, but give the VPN service providers control over the traffic content. The data is encrypted from the sender and to the VPN provider, but not further. This is inherently limiting the provided protection, because data is not only accessible by the intended receivers, but also by the VPN provider and all subsequent nodes in the path to the receiver.

Controlled admission and differentiated access to applications and services requires configuration of complex firewall rules with authentication mechanisms. This type of maintenance is only achievable by skilled users, and updating firewall rules for a roaming device is likely an endless task, because the circumstances are constantly changing.

Using services and applications in different networks simultaneously can require all of the efforts above repeated for each network, device and application. It is unlikely that anyone is determined enough to go through that for more than a few devices.

Some of the required efforts above can be impossible for end users to perform if they do not have administrative privileges on the network equipment, e.g., opening ports on routers. Our goal is to realize the user desires described above with minimal configuration efforts and need to modify equipment. We aim to go from complex configuration tasks which require expert knowledge to close to zero configuration requirements.

3. Related work

Several systems for device communication, sharing, and collaboration exist, but none of them are able to provide a truly ubiquitous experience. Some systems, such as Turtle [4], OneSwarm [5], Tarzan [6] and Tangler [7], only address anonymity and privacy issues, while others, such as WASTE [8], are limited to a specific application domain, e.g., file sharing. OpenVPN [9] and Mobile IP [10,11] have considerable setup and management efforts and are vulnerable, because of their single point of failure. The most closely related systems to DevCom use overlays and introduce a virtual network between the applications and the physical network. This includes GroupVPN, SocialVPN [12], TinCan [13], Network 2 Network [14], P2PVPN [15], Everywhere Local Area Network [16] and the Unmanaged Internet Protocol [2].

GroupVPN and Social VPN are two systems built on top of a framework called IP over P2P (IPOP) [17,18], which have later evolved to TinCan. Neither of the systems support multiple simultaneous networks in their current versions.

Network 2 Network (N2N) provides multiple simultaneous networks which are called communities. The N2N communities are configured manually and statically, and dedicated super peers are used for bootstrapping and NAT penetration. As pointed out above, manual configuration is a potential security weakness, and assigning dedicated servers which are always available is unrealistic from an end-user perspective. N2N uses pre-shared keys which makes eviction of devices from N2N communities impossible.

P2PVPN relies on centralized BitTorrent trackers for bootstrapping, requires manual configuration of IP addresses, and does not penetrate NAT. The current design and implementation does not allow multiple simultaneous networks, and a final drawback we found when testing P2PVPN, is that it drops packets when packet sizes are large.

Everywhere Local Area network (ELA) uses both UDP and TCP for tunneling in order to support different types of NAT. A differentiation is made between devices using both UDP and TCP to communicate (core nodes) and devices using only TCP (edge nodes). Bootstrapping is done manually in ELA by providing an IP address of a device already in the network. Data packets are supposedly encrypted, but the details of how this is done are not disclosed.

The Unmanaged Internet Protocol (UIP) is part of a larger project called Unmanaged Internet Architecture (UIA) [19]. UIP allows several users per device, but it does not support different coexisting networks. Different network names, known as personal groups, are possible, but they are in reality aliases for the same network. This means that applications and services cannot be restricted to certain networks. Another drawback of UIP is that it does not have TCP fallback when UDP hole punching[1] fails. This can result in unreachable devices, e.g., in presence of symmetric NAT.

The above review reveals that the most important feature, i.e., support for multiple simultaneous networks, is only supported by N2N. It is probably possible to extend some of the related work, e.g., TinCan and UIP, to have multiple IP addresses, but it is unclear whether their design allows the deterministic assignment novel to DevCom. In particular, DevCom uses a network part mapped to a community name, and a unique, static host part mapped to a public key identifier. DevCom's IP address assignment enables multiple, unique, persistent IP addresses and differentiated access to resources, based on network/community membership. The second most important feature, i.e., complete self-configuration, is only provided by TinCan and UIP. None of the related systems provide both features combined.

Table 1
Trust related tasks.

Task	Cryptographic keys	Component
Establish trust	Public RSA key exchange	User
Sign and decrypt control traffic	Private RSA keys	Overlay Manager
Verify and encrypt control traffic	Public RSA keys	Overlay Manager
Encrypt and decrypt data traffic	AES symmetric key	Data Manager

4. Design

The overall goal of this work is to provide end-users with a user-friendly and trustworthy platform to communicate, share and collaborate among small groups of devices, e.g., home devices, work devices, etc. This should be achieved for existing applications and with existing network solutions, and without the need to request the user to perform modifications or cumbersome configurations. In the following, we briefly motivate for the major design decisions to achieve these goals, before we present the following subsections the detailed design of device communities, the architecture of DevCom, how it internally works during practical use, and the most important implementation aspects.

Our aim to achieve the highest possible degree of trustworthiness for realistic settings, i.e., using any kind of computing device over the unprotected Internet, implies two major design decisions. First, the system must be entirely independent of 3rd parties such that only devices the user personally trusts are involved. Second, cryptographic functions with a high security level must be applied to protect all data that is exchanged among the trusted devices. Table 1 gives a brief overview over the tasks related to trust, the cryptographic methods used for these tasks, and the responsible DevCom components. To achieve independence of 3rd parties and user-friendliness at the same time requires a self-organizing system with close to zero configuration effort for the user.

One major task of DevCom is to manage the membership information and maintain application sessions. Device mobility and churn introduce a challenge for this task since the IP addresses of devices[2] change in an uncontrolled and unpredictable manner. The fundamental problem that IP addresses combine identification and location in one address is circumvented in DevCom by adding a virtual IP layer on top of the existing, unmodified Layer 3, i.e., the native IP layer. The virtual community addresses are static such that applications always see the same IP addresses, even when the underlying, physical device addresses change, e.g., due to handover from WiFi to 4G. In order to achieve maximum resilience for device communities against churn we propose a fully decentralized solution in which all devices maintain a list of all devices they have a relationship with, i.e., all devices they trust in at least one device community. Furthermore, all devices propagate device community membership changes to other members. This implies that all devices are equal in terms of trust and responsibilities regarding

[1] UDP hole punching is a technique for NAT traversal.

[2] Depending on the context, the IP address of a device is later in this paper also called physical address or location.

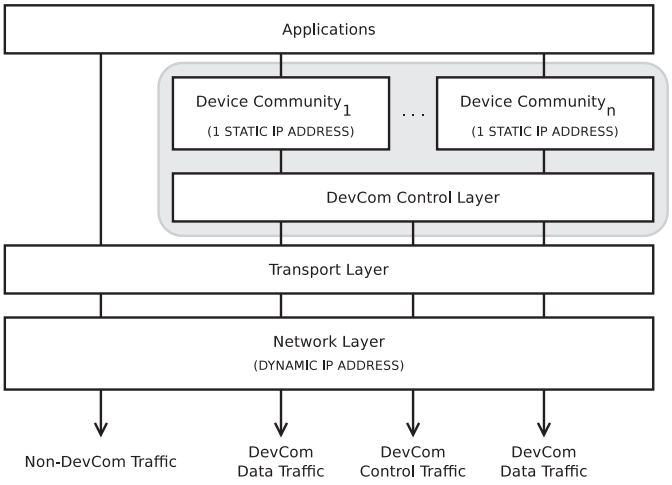


Fig. 1. Layers of the DevCom stack.

device community management. For a device that has been turned off for a longer time the membership information it hosts might be outdated to a large extent, since many of the other devices might have obtained new IP addresses. However, a single valid IP address of another device community member is sufficient to synchronize the membership information and again gain full connectivity for all device community members.

These core design decisions lead to the networking stack in which applications can use either the existing transport layer services or the services of one or more device communities which provide persistent connections, security and protect privacy (see Fig. 1). Device communities are built on top of a DevCom control layer to maintain these overlay networks over the existing Internet. Fig. 1 also indicates the relationship between the physical IP address of the device which can change and the static virtual IP addresses of device communities.

4.1. Device communities

The fundamental feature that DevCom attempts to provide is user-friendly and trustworthy communication, sharing, and collaboration among any potential devices users desire. This requires a novel solution to enable a device to be member of multiple device communities at the same time and have simultaneous access to the resources their members provide. In existing network systems such as VPNs, a device can only be a member of one network at a time, inhibiting the device from concurrently using, e.g., services at work (remote desktop), from friends (file sharing) and at home (printer). The core idea to isolate data between different device communities is to assign multiple, permanent, virtual IP addresses to each DevCom device, i.e., one address for each device community a device is a member of.

Another important requirement that has to be resolved is user-friendly configuration. In contrast to other systems where users have to manually define IP ranges, DevCom does this automatically in order to not burden the user and to avoid human errors. Furthermore, current addressing techniques can assign changing IP addresses to mobile devices depending on their physical location, and even to stationary devices that use DHCP. Dynamic addresses make it difficult for users to locate their devices to access files, peripherals and services, and more importantly, break all open connections. When a device changes its address and the open connections break, it effectively stops all file transfers, remote desktop sessions, audio/video streams etc. This is especially problematic for mobile devices which frequently change between cellular and

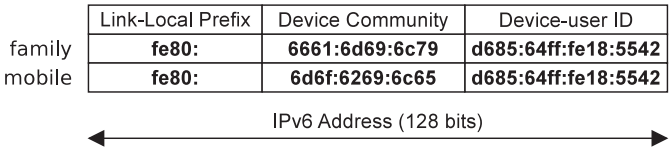


Fig. 2. Two example device community addresses.

Wi-Fi networks. DevCom must provide static addresses to ensure that all devices in all device communities are easily identified, and to enable persistent connections.

The way DevCom achieves static addresses is with a self-configuration technique that is more user-friendly and less prone to address collisions and misconfiguration than manual efforts. It combines a known prefix with the device community names and cryptographic public keys to create one static and unique address for each device community. Link-local IPv6 addresses are used instead of IPv4 addresses, because the IPv4 address space is exhausted and it is impossible to generate an IPv4 address and guarantee that it is not in use. As such, it is also impossible to guarantee that traffic for a device is routed to the DevCom virtual network interface instead of the physical network interface.

Fig. 2 shows how DevCom separates the device community addresses into three parts: the link-local prefix, the device community name, and the device-user identification. In contrast to, e.g. Distributed Hash Tables (DHTs) and other P2P systems that use proprietary addressing schemes, DevCom uses IP addresses that work with existing Internet applications. The following algorithm is used to construct the IP addresses.

The first 16 bits are the IPv6 link-local prefix. This prefix, fe80, is defined by the IPv6 standard and is intended for communication within the segment of a local network or a point-to-point connection. It enables DevCom to provide a LAN equivalent network over any underlying network. Using the fe80 prefix minimizes the chances of address collisions, because link-local addresses are not globally routed on the Internet.

The following 48 bits comprise the device community name represented in hexadecimal. Typical device community names might be `work`, `mobile` and `family`, but this is entirely up to the users to decide. The names do not need to be globally unique, meaning that two independent users can both have, e.g., a `home` device community. The result is that all devices in a device community have the same $16 + 48 = 64$ bits network identifier prefix.

The last 64 bits represent the unique identity. DevCom uses public-key cryptography in order to provide security features, and the self-configuration mechanism uses a fingerprint of the user's public key as a unique identifier on each device. The unique $16 + 48 + 64 = 128$ bit addresses ensure that DevCom only delivers data to the device community members it is intended for.

Another important requirement is trustworthy communication. It is vital that privacy and security is maintained with state-of-the-art cryptography algorithms, and that devices can be evicted from the different networks if they are misplaced.

Fig. 3 illustrates how this addressing concept is used to separate traffic to different device communities from each other and from non-DevCom data traffic. The six steps of Fig. 3 show how any network application can send data in a trustworthy fashion using DevCom. (1) An application is sending data using the normal network stack, completely unaware of DevCom. (2) Standard prefix routing ensures that all packets are delivered by the operating system to the virtual network interface, separating device community traffic from regular Internet traffic. (3) The data packets are picked up by DevCom which determines the current physical address of the receiving device and encrypts the packets using a symmetric key only known to the sending and receiving DevCom devices. (4)

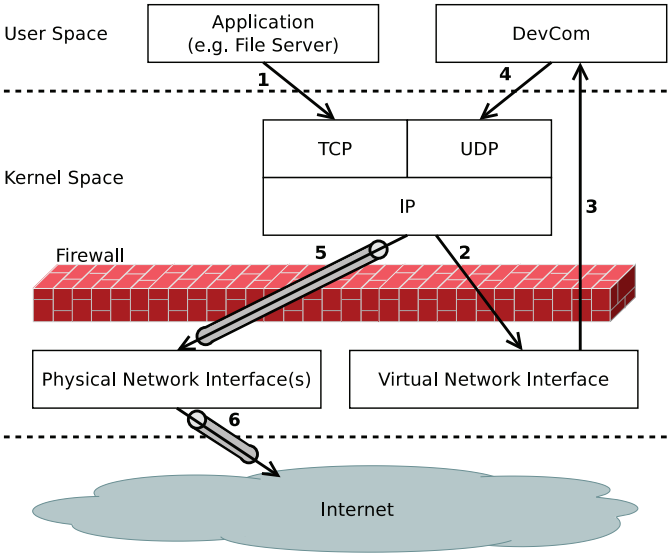


Fig. 3. Example of DevCom data flow.

The data packets are encapsulated in UDP/IP and (5) tunneled to the receiving device using the physical network interface. (6) The tunneled data packets are sent via the Internet, and delivered to the receiving DevCom device which reverses the process.

In the receiving device, the operating system kernel delivers all tunneled packets to DevCom, because it is bound to the outer packet's destination port. DevCom uses the symmetric key corresponding to the current session to decrypt the tunneled packet and deliver it back to the operating system. The operating system recognizes that the virtual device community address is a local address and delivers the data to the application bound to the inner packet's destination port.

The six steps guarantee that data is only going to the intended recipient and that routers and other potential, intermediate devices only see encrypted data. If the user wants additional security, e.g., as part of a security-in-depth approach or because an application does not have its own authentication mechanisms, it is possible to use regular firewall software to deny access to applications per device community. How this is done is explained in more detail in Section 4.3.5.

4.2. Architecture

The DevCom architecture consists of four components, shown in Fig. 4. The components are present in all devices and perform the same tasks.

- The Virtual Interface Manager is responsible for the virtual network interface by assigning the virtual device community addresses.
- The Key Manager is responsible for key management, i.e., creating symmetric and asymmetric keys used to ensure authentication, confidentiality and integrity.
- The Overlay Manager is responsible for membership management and to maintain the control channels to the members. It manages the decentralized virtual overlay by sending and receiving control messages via these channels.
- The Data Manager is responsible for the data channels, including tunneling, encrypting, and decrypting data packets.

The Overlay Manager is the main component of DevCom, responsible for maintaining the device communities by managing control channels between members. Control channels are dedicated TCP connections between community members used to send

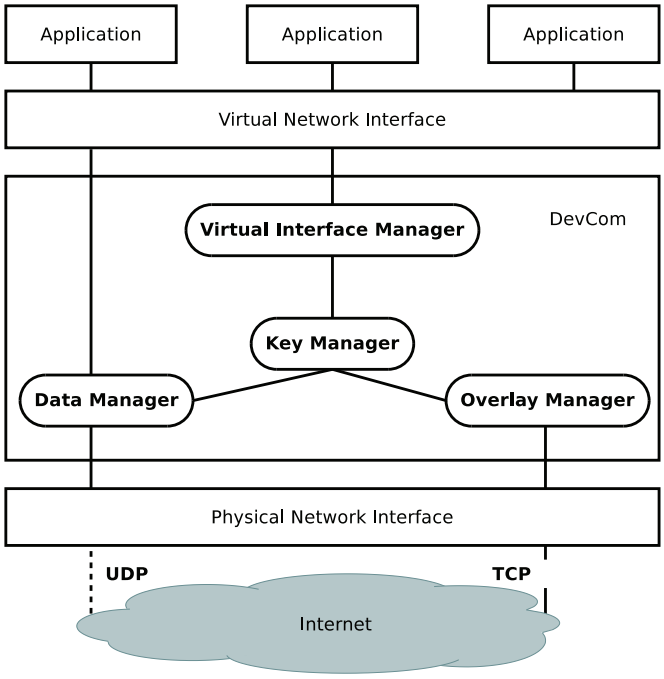


Fig. 4. DevCom architecture.

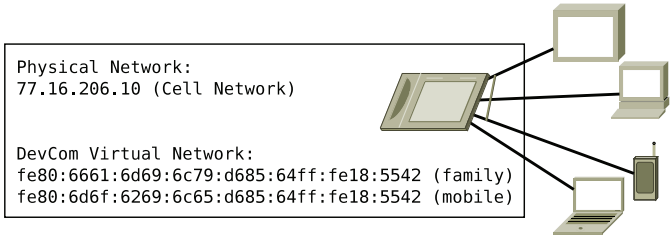


Fig. 5. Example of control channels and addresses.

and receive control messages related to device community functions, such as trusting new members. To avoid relying on third party service providers it is important that the Overlay Manager is self-organizing and decentralized. Fig. 5 shows an example with five devices that are in two device communities, i.e., `family` and `mobile`, and for the tablet computer its physical IP address and two virtual IP addresses for the device communities. The Overlay Manager on the tablet computer maintains control channels to all device community members, but only one control channel is established for each physical address, even if two devices trust each other in more than one device community. The control channels are made using any currently available Internet connection, meaning that it automatically switches on devices with, e.g., Wi-Fi and 3G connectivity. Changing physical network connectivity does not affect the self-configured, virtual device community addresses.

The Overlay Manager maintains mappings between the current physical IP addresses (locations) of the other devices and their virtual device community addresses (identities). The mappings are updated each time a device connects to a device community and authenticates itself, e.g., in roaming scenarios when a device switches from Wi-Fi to 3G connectivity. Historical records of physical IP addresses are stored in the Overlay Manager in order to assist creating the control channels. When a device wants to connect to a device community, the Overlay Manager attempts to locate devices at the different physical addresses that have been used in the past, starting with the most recent addresses.

Message Type (1 byte)	Community (6 bytes)	Fingerprint (8 bytes)	Extra Information (Different sizes)	Signature (512 bytes)
T(rust)	WORK	12345678	PUBLIC KEY	ABC[...]XYZ
D(istrust)	MOBILE	48616E73		123[...]789
J(oin)	ALL	13374242	SYMMETRIC KEY	111[...]999
L(ocations)	HOME	23122012	PHYSICAL ADDRESS	AAA[...]ZZZ
S(ync)	WORK	48616E73		ABC[...]789
P(acket)	MOBILE	13374242	PACKET	ABC[...]789

Encrypted

Signed

Fig. 6. List of DevCom control messages.

The control message payloads are encrypted with the receivers public key to prevent eavesdropping and ensure privacy, and signed with the senders private key, authenticating the sender and verifying the integrity of the message. Standard asymmetric cryptography is used and the 4096 bit RSA keys are provided by the Key Manager. The cryptography provides integrity protection of the control messages and authentication of the devices. Fig. 6 shows the different control messages used by the Overlay Manager to manage the device communities.

T(rust) is sent to instruct a device to accept another device into a device community by storing the new device's public key. D(istrust) is sent to instruct a device to evict another device from a device community by deleting the identified device's public key. J(oin) is sent to establish a control channel including the exchange a symmetric session key. L(ocations) is sent to inform a device about another device's IP addresses. S(ync) is sent to initiate a synchronization of a device community, i.e., start exchanging T(rust) and L(ocations) messages. P(acket) is used to send data packets over the control channel if the data channel is unavailable, e.g., because of UDP filtering.

Control channels are only used for control messages. To send application data so-called data channels need to be established by the Data Manager. This separation of control and data traffic is done because early measurements show that using UDP instead of TCP tunnels increases throughput by 27% and decreases average latency by 23%. It is also documented that TCP tunnels can degrade performance when the end-to-end TCP and the tunnel TCP interfere with each other [20]. Data packets are therefore tunneled by the Data Manager using UDP, with a fallback to TCP using the P(acket) control message if UDP is blocked. The reasoning behind the fallback to TCP is that a suboptimal connection is better than no connection.

The Data Manager is responsible for maintaining data channels to all devices in a device community, separate from the control channels, shown in Fig. 1. The Data Manager must be efficient, but without compromising privacy. For this reason, it uses 256 bit Advanced Encryption Standard (AES) symmetric keys, retrieved from the Key Manager, to encrypt and decrypt data traffic. Cipher Block Chaining (CBC) mode is used. The encryption preserves privacy while the symmetric AES cipher is faster than the asymmetric RSA cipher to improve efficiency. The AES keys are exchanged over the control channel to guarantee security and reliability.

The Virtual Interface Manager is responsible for maintaining the virtual network interface, i.e., automatically creating and assigning virtual IP addresses. It must separate the identification (virtual interface) and location (physical interface) of a device and enable persistent connections when a device changes physical networks. The Virtual Interface Manager achieves this by creating permanent addresses using the self-configuration algorithm described above and associating the addresses to a virtual network interface placed between the applications and the physical network interfaces.

The Key Manager is responsible for creating and maintaining the asymmetric RSA keys and the symmetric AES keys used by the Overlay Manager and Data Manager, respectively.

4.3. Practical use of DevCom

4.3.1. Key management

The RSA public and private key-pair is automatically created the first time a user starts DevCom on a device. The asymmetric key pair is unique for each user on a device, and two users owning a device together can be members of different device communities on that device. However, several users cannot run DevCom simultaneously, but have access to their device communities when they have exclusive access to the shared device.

The private and public keys are never handled by users. A user only has to select which devices to trust in which device communities and all necessary information about the new device community member is automatically propagated by the Overlay Manager to all existing members. Similarly, in a situation where a device is lost or misplaced, the user only has to select that device and the Overlay Manager takes care of removing the corresponding public key and closing all communication with the device. The eviction information is also distributed to all the other members of the device community. It is possible to evict a device from one device community, while continuing to trust it in another.

4.3.2. Joining a device community

When a user wants a device to initially join a device community, two tasks need to be performed: (1) provision of a physical address of a device already in the device community, and (2) a two-way, public key exchange with that device. The required information can be exchanged between the two devices in one of the following four ways.

Service discovery protocols such as Multicast DNS (mDNS) [21] and DNS-based Service Discovery (DNS-SD) [22] can be used to discover other devices nearby. To trust a device for a given device community, the user simply selects the desired announcement in the user interface.

Near field communication (NFC) is a radio communication standard where devices have to be in close proximity to exchange data. Mobile devices can use NFC to share the required information.

Quick response (QR) codes are two-dimensional bar codes that can encode and present the required information visually. Personal devices can generate QR codes on demand, but they can only be used when the devices have cameras.

Invitation files can be generated on-demand to contain the required information. The files can be exchanged via memory sticks, e-mail or instant messaging and users only have to click on the files to join a device community.

4.3.3. Connecting to a device community

The established control channels break if a device is roaming or it is turned off for some time, because it receives a new physical IP address. In such cases, the device needs to reconnect to the device communities it is member of by re-establishing the control channels. When this happens, the mutual trust is already established through the two way public key exchange during the join operation. Therefore, the connecting device only needs a valid physical address of one of the other device community members.

DevCom uses the last valid membership information to automatically probe for devices at the different locations they have been used in the past, starting with the most recent. If the device is connecting after a short switch of networks, e.g., moving from a cabled to a wireless network or from Wi-Fi to 3G, the location of at least one of the other devices is presumably unchanged. It is highly unlikely that *all* devices in a device community get new physical

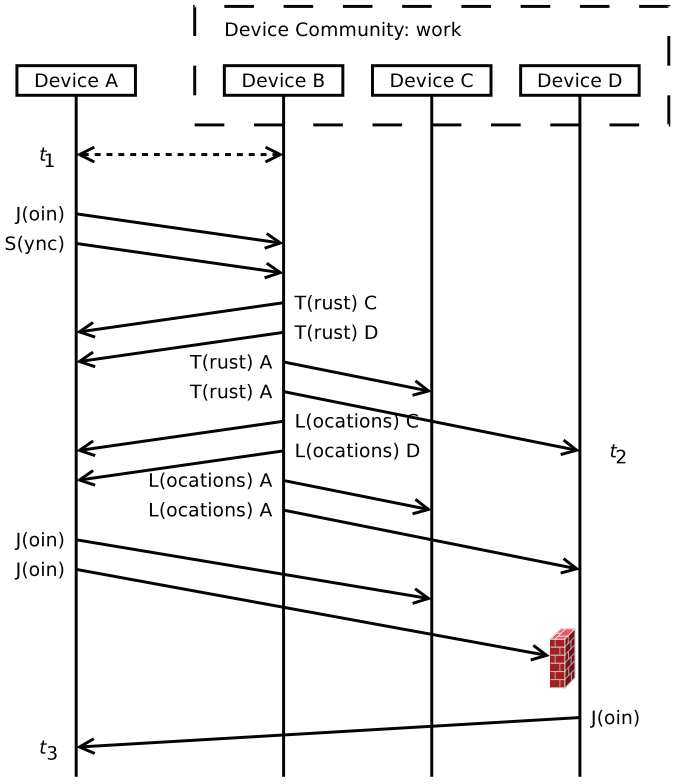


Fig. 7. Device community connection example.

addresses in the few seconds it takes to switch networks. However, if more time has passed, e.g., because a device has been turned off for a few days, it is more probable that all the other devices have new physical addresses. In these cases DevCom attempts to find the other devices using their historical addresses. Records of historical addresses are always kept in order to assist this type of probing. Authentication is still provided by the asymmetric keys, so no malicious third-party device can deceive the connecting device by obtaining the historic address of another device and pretend to be it.

It is important to point out that only one existing member needs to be found. Any member of a device community can be used to synchronize membership information, because all members maintain locally complete knowledge about the device community memberships, and no special role, like a guard member, is used. Synchronization control messages ensure that the connecting device gets information about all the other members, and vice versa. When only one device is available it becomes a single point of failure, but when more members are available, the likelihood of locating a member increases and the load is distributed so it does not become a performance bottleneck. Thus, the probability of finding a trusted device increases with the size of the device community. However, if no device can be located, the user has to resort to one of the initial joining techniques described in Section 4.3.2.

A device can start communication with members it has established control channels to while the connection procedure continues to the remaining devices, minimizing the perceptibility for the user. Similarly, the connection procedure is transparent to the user if no applications are actively communicating to the device community.

Fig. 7 illustrates an example scenario where Device A is about to join and connect to a device community (`work`) for the very first time. One caveat is that Device D has a firewall that does not allow incoming connections.

At time t_1 Device A and Device B are sharing public keys and physical addresses, using any of the methods mentioned above. After mutual trust is established, Device A sends a J(oin) message to Device B in order to create the control and data channels. In this J(oin) message, it includes a pseudo-randomly created AES session key which they use to encrypt and decrypt application data.

After the control and data channels are created, Device A sends a S(ync) message to Device B with the consequence that Device B informs Devices A, C and D about each other, using T(rust) and L(ocations) messages. The T(rust) messages contain the public keys to trust, and the L(ocations) messages contain the physical addresses to connect to.

At time t_2 all four devices trust each other and are considered members of the "`work`" device community, even if not all devices are currently connected. Device A sends a J(oin) message to Device C, and attempts to send a J(oin) message to Device D, but is rejected in the latter attempt, because of the firewall. However, when Device D gets the L(ocations) message from Device B and realizes that it does not have a connection with Device A, it establishes a reverse connection by sending a J(oin) message to Device A.[3] At time t_3 all the devices are connected with a full-mesh topology.

A similar pattern of messages is exchanged in all scenarios when a device connects to a device community, e.g. when a device regains power after an outage, or when switching between physical networks. The scalability of DevCom is limited by the connection procedure, i.e., by the time it takes to connect to all device community members. Currently, the number of packets sent during the connection procedure converges towards 5 packets per device community member, where, e.g., connecting to a device community with 100 members requires 492 packets to be sent.

4.3.4. Persistent connections

When a non-DevCom device changes its IP address, e.g., when moving from a cabled to a wireless network or from Wi-Fi to 3G, all open connections break. Remote desktop sessions halt, file transfers stop, and games and audio/video streams freeze. The reason is that an IP address incorporates both identification and location of a device.

With DevCom, persistent connections are not only possible, but automatic. Applications use the virtual device community addresses and not the changing physical addresses, decoupling the identification (device community address) from the location (physical address). When a physical address changes, the device has to connect again to the device community. This is invisible to the application except a short term increase in delay. This means that a user commuting home from work while streaming a movie from a friend does not have to restart the video playback or reconnect the player application when network changes occur. The seamlessness of the experience depends on a number of factors, such as the size of the buffer and how fast the physical network switch occurs.

4.3.5. Firewalls and NAT

Firewalls and NAT can limit connections between devices and in some cases completely block certain applications and services. DevCom uses existing techniques in order to attempt to provide connections between trusted device community members that would otherwise not be possible. This does not make DevCom devices more vulnerable, because regular Internet traffic, including potential malicious traffic, is still blocked as intended. NAT penetration is performed using standard techniques, such as reverse connections and relaying. Relaying devices are unable to read or tamper with the data, because of the strong cryptography.

Firewalls can have many ways of limiting traffic. Four filtering techniques that DevCom attempts to bypass are deep packet

[3] More information on firewall and NAT related issues are given in Section 4.3.5.

Table 2
Example of firewall access rules.

Device community	Application/Service	Rule	Direction
`mobile`	Web-cam streaming	DENY	IN
`mobile`	File sharing	DENY	IN & OUT
`family`	Remote desktop	DENY	IN

inspection, port blocking, UDP blocking and host blocking. Deep packet inspection and blocking traffic based on data content or application protocol (e.g., BitTorrent) is made significantly harder, because all data including protocol headers is encrypted. Furthermore, port blocking can in many situations be circumvented, because the data traffic is tunneled and not delivered to the actual service port, such as port 22 for ssh, but to the DevCom port. UDP blocking can also be circumvented, because DevCom automatically falls back to TCP if UDP is blocked. Lastly, host blocking can be circumvented by relaying traffic through other device community members using the P(acket) control message. All device community members accept P(acket) control messages containing encapsulated (data or control) messages. If a P(acket) message is not indented for a receiving member, it will forward it. For example, if Devices A and D in Fig. 7 are unable to communicate directly, even after attempting the reverse connection, Device B or C can be used as proxies.

However, firewalls can also be of benefit to DevCom. If needed, it is possible to restrict applications and services to certain device communities using common application firewalls. For example, a tablet computer can have a web-cam service announced and available to `family` devices, but not to other `mobile` devices, which remain unaware of this service and unable to connect to it. Existing firewalls with graphical user interfaces and wizards make it easy for users to pick applications from a list and choose which device communities to allow it in. One set of example rules[4] to differentiate access privileges is shown in Table 2.

4.4. Implementation

The DevCom prototype implementation consists of 3686 lines of C code[5] and is compiled for the Intel and ARM architectures. It uses the TUN/TAP driver [23,24] to create the virtual network interfaces and the Avahi library [25] for device discovery and trust exchange. The OpenSSL library [26] is used for all cryptography functions. This includes generation of the asymmetric key pair the first time a user starts DevCom and encryption and decryption of data using the session keys.

While the DevCom design is fully operating system agnostic, the current implementation has only been tested on Linux. A partial test has confirmed that Android, which is based on the Linux-kernel, is supported if the TUN/TAP kernel module is installed. The module is available for most Android devices [27], but requires root privileges to install if it is not pre-installed by the device vendor. iOS devices are not supported, because they use the Objective-C programming language.

One important implementation decision worth mentioning is choosing to copy packet data between the virtual network interface and the physical network interfaces instead of zero-copying the packet data using the splice() Linux system call [28]. Contrary to our expectations, early measurements show that reading and writing instead of zero-copying increases average throughput by 22% and decreases average latency by 3%. Colloquial sources on the world wide web experience similar decrease in performance when splice() is used with small buffer sizes. This means that avoiding zero-copying makes the DevCom implementation more portable and more efficient for all applications using it.

Time constraints have prevented the implementation of NAT traversal and firewall penetration through relaying, leaving reverse connections as the only employed penetration technique.

5. Evaluation

The overall goal of DevCom is to move todays state-of-the-art in ubiquitous computing one step closer towards Mark Weiser's vision to "make computing an integral, invisible part of the way people live their lives". With DevCom, computing is still visible for users, but the core properties of DevCom, i.e., user-friendliness, security, privacy protection, and minimal overhead to communicate among devices, reduce the necessary effort of users to merge all their devices in a trustworthy device community. DevCom provides these properties in the very complex and challenging environment of todays' Internet with wireless and mobile devices, large variations in computing power of devices, large variations in available bandwidth, and devices behind firewalls and NATs. Therefore, it is also a challenge to properly evaluate DevCom. The major aspects to be evaluated are related to functionality, performance, and user-friendliness. Only security and privacy protection is not evaluated in this paper since it is achieved with existing state-of-the-art solutions.

The evaluation of the functionality aspect comprises two parts, first the analysis of which types of applications can be used on top of DevCom. Second, the analysis of whether the system is able to maintain connectivity among device community members in the case of mobility, NATs and blocking firewalls, i.e., if application sessions can be maintained in such situations. Thus, to evaluate the second part of the DevCom functionality, we perform two experiments in real-life environments.

The performance evaluation of DevCom includes detailed measurements of the overhead this extra layer between the physical network and applications introduces, using the metrics latency, throughput, and computational overhead. To achieve accurate results many experiments are performed in a fully controlled lab environment. In order to understand the implications of the overhead, we do not only measure the overhead of DevCom itself. Additionally, we perform the same experiments with nearly all related works in order to compare DevCom performance with the performance of GroupVPN, N2N, P2PVPN, SocialVPN, and UIP. Finally, we perform an empirical experiment to compare the quality of experience (QoE) of a first-person shooter game with and without DevCom.

This leads us to design and perform five experiments, outlined in Table 3. In addition, the user friendliness of DevCom is studied from the viewpoint of novice users, intermediate users, and application developers.

The devices used in the experiments and their specifications are listed in Table 4. In order to eliminate underlying factors such as network conditions and isolate the relevant metrics, transparent testbeds consisting of two identical devices connected to the same switch are used. The desktop devices represent high-end personal devices, while the laptop devices are more representative of typical, everyday devices. The handheld devices represent both legacy devices, such as the Apple iPhone 3GS, and more current, but resource constrained devices, such as set-top boxes and network-attached storage (NAS) devices. Similarly, the different network connections represent the whole range of bandwidth typically available to consumers, from 54 Mbps to 1 Gbps connections.

Benchmarks using desktop computers with excellent network connectivity are especially suitable to highlight the overhead of

[4] Created by *Graphical Interface to Uncomplicated Firewall (gufw)*.
[5] Determined using SLOCCount by David A. Wheeler.

Table 3
The five conducted experiments.

Aspect	Experiment	Description
Functionality	Off-the-shelf applications	Tests existing applications for DevCom compliance.
	Self-configuration	Verifies DevCom functionality and practical applicability.
Performance	Micro-benchmarks	Measures latency, throughput and CPU utilization.
	Quality of experience	Compares DevCom with native networking in an A/B test.
	Connecting time	Investigates the impact of changing network connectivity.

Table 4
Device hardware specifications.

	Desktops	Laptops	Handhelds
Brand	HP Compaq	HP EliteBook 8530p	Nokia N900
Processor	Intel Core i7 2.93 GHz × 8	Intel Core 2 Duo 2.53 GHz × 2	ARM Cortex-A8 600 MHz × 1
Memory	8 GB	4 GB	256 MB

Table 5
Testbed configurations.

Testbed	Devices	Network connectivity
A	2 × Desktops	1 Gbps, wired
B	2 × Desktops	100 Mbps, wired
C	2 × Laptops	100 Mbps, wired
D	2 × Laptops	54 Mbps, wireless
E	2 × Handhelds	54 Mbps, wireless

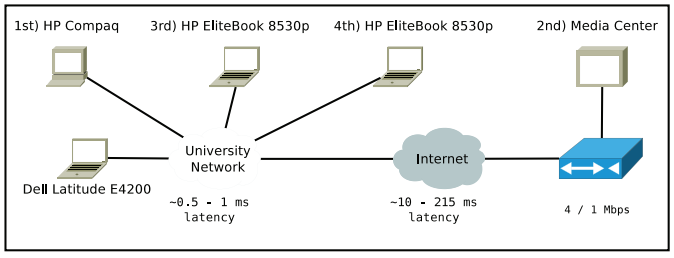


Fig. 8. Testbed F, used in the connecting time experiment.

the different network systems compared to direct network connections. Benchmarks using handheld devices with poor network connectivity allow investigation of how DevCom performs in the worst possible use-cases.

The testbed configurations used in the experiments are shown in Table 5 and Fig. 8. Testbed A is used in the off-the-shelf applications experiment and the quality of experience experiment. Testbeds A through E are used in the micro-benchmark experiments. Finally, testbed F is used in the connecting time experiment. The HP Compaq desktop and HP EliteBook laptops in testbed F are the same machines as in Table 4, while the Dell Latitude and Media Center machines are other machines with specifications comparable to, but less powerful than, the laptops. The network conditions in testbed F are varying corresponding to the background traffic. The end-to-end latency changes internally in the university network and over the Internet. The media center is connected to the Internet via a cable modem line with 4 Mbps downlink and 1 Mbps uplink.

5.1. Functionality

5.1.1. Off-the-shelf applications

In order to demonstrate that DevCom works with all types of existing network applications and services, it has been tested with the large variation of tasks listed in Table 6. The list is not exhaustive, but is intended to represent many different types of network applications. The applications have been used for their regular tasks for up to one hour each. Most time is spent using the remote desktop, games and video streaming applications, and least time is spent using the network diagnostic tools.

Table 6
Tested off-the-shelf applications working with DevCom.

Type	Name
File and printer sharing	Samba
Remote desktop	Remmina/Vino
HD video streaming	VLC
Games	OpenTTD and Quake 3
File transfer	FileZilla/vsftpd
Backup	rsync
Secure shell	ssh
File sharing	sshfs
Network diagnostic tools	tcpdump, nmap and netstat

The tests reveal that all the applications work without modification. DevCom is able to handle many different types of applications and network traffic patterns without introducing any annoyance to the user in form of, e.g., configuration efforts or noticeable quality degradation.

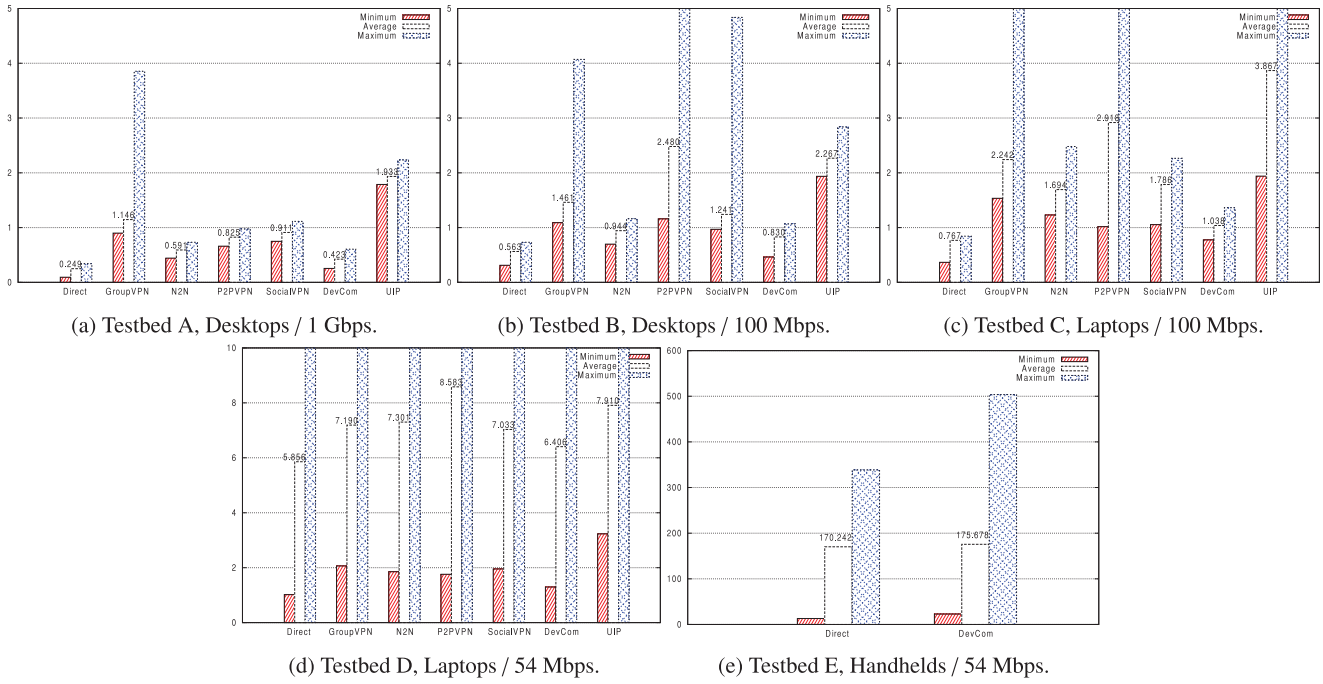
5.1.2. Self-configuration

This experiment intends to get insight into the practical use of DevCom functionality. Eight distributed devices with varying specifications form a device community, and users (informatics students) test how well the DevCom properties work. The device community is operational for several days, but the actual use of applications are, as in real life, more ad-hoc. The main test involves streaming audio using the VLC media player from a device that switches networks once every minute. A firewall is blocking incoming UDP connections to the streaming server and one residential device is behind a NAT box.

The experiment verifies that the self-configuration works. The public and private key-pair is created the first time the users start DevCom, the permanent addresses are generated, and all member information including the cryptographic keys are propagated within the device community. Furthermore, the NAT and firewall penetration also works as expected, and the users are able to access devices in spite of UDP and port blocking firewalls. The audio streaming continues without user intervention after the streaming server switches networks, but with a delay corresponding to the downtime of the physical network, because of no buffering. Lastly, no devices have problems connecting to the device community, because at least one device is always on-line.

5.2. Performance

Micro-benchmarks measuring application to application latency and throughput, as well as CPU utilization, provide a performance comparison between direct network connections (expressed as “Direct” in the following graphs), DevCom, and most of the related work; and determine their overhead. Direct network connections in the following sections means the use of the different applications and services without DevCom or the related work running,



(a) Testbed A, Desktops / 1 Gbps. (b) Testbed B, Desktops / 100 Mbps. (c) Testbed C, Laptops / 100 Mbps.

(d) Testbed D, Laptops / 54 Mbps. (e) Testbed E, Handhelds / 54 Mbps.

Fig. 9. Latency results. The *y*-axsis is in milliseconds.

i.e., corresponding to the leftmost arrow in Fig. 1. The different applications and services used in the experiments have different characteristics, like TCP or UDP, but these are identical when running on top of the direct connections and on top of, e.g., DevCom. The following results show that the performance penalty of providing the flexibility of device communities and adding security features is in most cases negligible, except in the throughput experiment using testbed A.

5.2.1. Latency

High latency and noticeable delay can negatively affect the user experience of real-time applications, such as first-person shooter games, video conferencing applications and remote desktop services. Minimizing latency overhead is therefore important in order to support these types of applications.

Latency is in our experiments determined by using the ping tool and sending 1000 packets. The minimum, average and maximum round-trip times are shown in Fig. 9. The graphs are not normalized in order to better highlight the differences.

The results show that the average latency overhead of the different network systems is comparable, but DevCom has lower average latency than the related work, regardless of device hardware and connectivity. Some of the other systems also exhibit higher variance, but overall, the systems perform similarly. It seems that the latency is affected to some degree by the device hardware and network system, but mostly by the network connectivity. That network connectivity is the major factor impacting latency is an indication of low overhead.

The average latency overhead of the network systems is small when compared to a direct connection. For example, DevCom only increases the average latency with approximately 0.2 ms using testbed A. Furthermore, DevCom's average latency overhead is less than 0.6 ms when compared to direct communication using testbeds B, C and D. In testbed E, the average latency overhead of DevCom is approximately 5 ms, but this is only an increase of 3%, because the average latency using a direct connection is as high as 170 ms.

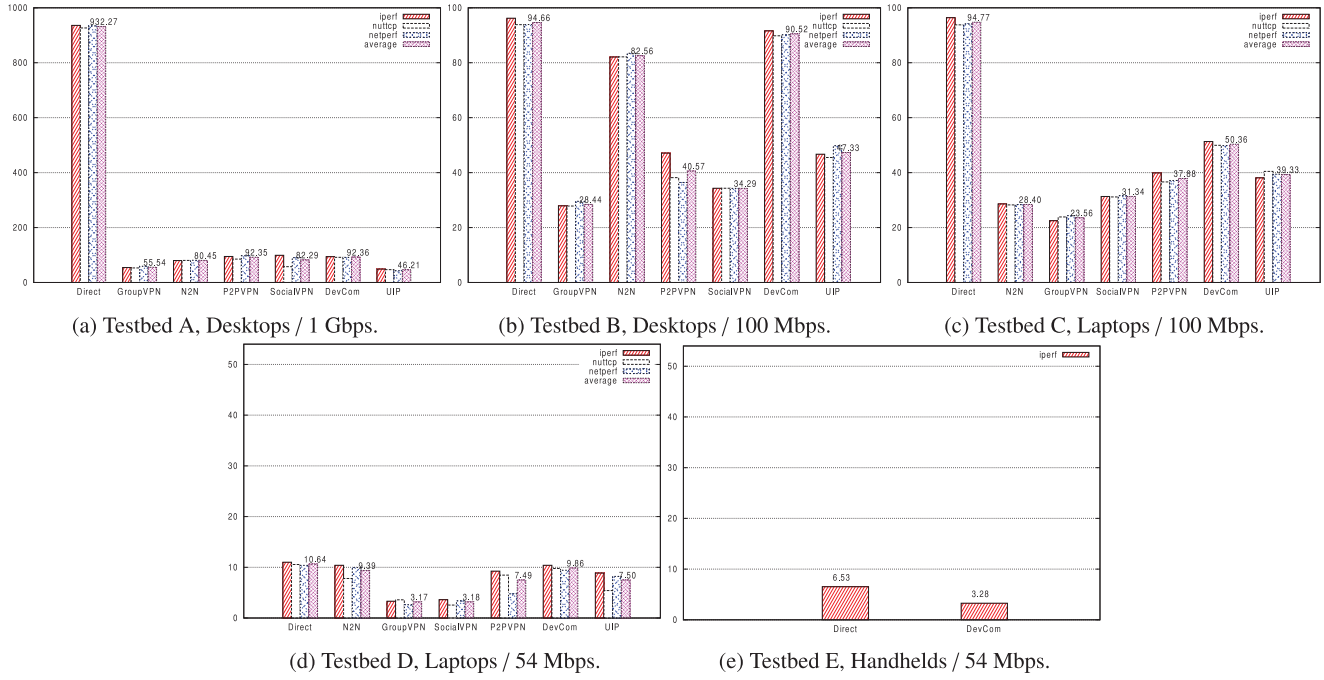
The latency experiments demonstrate that DevCom exhibits little latency overhead and that it therefore can allow real-time applications to be used without quality degradation.

5.2.2. Throughput

High throughput is important for applications such as video streaming and file sharing. In order to compare the throughput of the different network systems, as well as the baseline throughput of direct communication, three benchmarking tools are used, i.e., iperf, nuttcp and netperf. The results are shown in Fig. 10, together with the average result of the three tools. The graphs are not normalized, but show instead the maximum theoretical throughput of each testbed, e.g., 54 Mbps in testbeds D and E.

The results show that the throughput of all evaluated network systems is close to the throughput of direct connections on networks with limited bandwidth, regardless of device hardware. The bottleneck shifts from the network to the end systems when bandwidth increases, and in extreme cases with very high bandwidth, i.e., testbed A, none of the network systems achieve throughput close to the throughput of direct connections. In testbed A, DevCom and P2PVPN achieve only about 10% of the 932.27 Mbps throughput of a direct connection, and UIP and GroupVPN achieve a throughput of 46.21 Mbps and 55.64 Mbps. One possible bottleneck is the TUN/TAP driver that manages the virtual network interfaces, used by all evaluated network systems. If it is the bottleneck, it seems that it reaches its maximum throughput around 100 Mbps, but time constraints have prevented further research on the internals of this driver.

In testbed B, DevCom achieves a throughput of 90.52 Mbps and N2N 82.56 Mbps. These results are close to the 94.86 Mbps throughput of direct connections. All other systems achieve a substantially lower throughput, between 28.44 Mbps for GroupVPN and 47.33 Mbps for UIP. UIP, DevCom, and N2N achieve in testbed A and B approximately the same throughput, while GroupVPN, P2PVPN, and SocialVPN can benefit from the additional bandwidth in testbed A compared to testbed B and approximately double their throughput.



(a) Testbed A, Desktops / 1 Gbps. (b) Testbed B, Desktops / 100 Mbps. (c) Testbed C, Laptops / 100 Mbps.

(d) Testbed D, Laptops / 54 Mbps. (e) Testbed E, Handhelds / 54 Mbps.

Fig. 10. Throughput results. The *y*-axis is in Mbps.

The results from the experiments with the wireless network, i.e., testbed D and E, show that the throughput in all experiments, including direct connections, is below the available bandwidth. This is to be expected, because the attainable goodput is always lower than the maximum theoretical bandwidth of 802.11 networks. Inhibiting factors include Wi-Fi specific parameters such as slot time, distributed inter-frame spaces (DIFS) and short inter-frame space (SIFS), but also TCP/IP header overhead. The low throughput on the handheld devices, i.e., testbed E, might be contributed to the low CPU power.

Overall, the reduction in throughput that DevCom imposes is acceptable. It has the highest average throughput of the evaluated systems in all testbeds, which should allow bandwidth demanding applications to be used in a non-disruptive way.

5.2.3. CPU utilization

Low processing overhead is important in order to support a wide spectrum of computing devices, including low performance handheld devices, and to allow other applications to run on the same device. In order to measure CPU utilization, netcat (nc) is used to send data packets over TCP, while sar is used to measure the kernel space and user space CPU utilization. The difference between direct communication and the given network systems represents their processing overhead. Samples are gathered every second for 1000 s,[6] and the average results are shown in Fig. 11. The graphs are not normalized to 100% in order to better highlight the differences between the network systems in each testbed.

The results show that the CPU utilization varies with device hardware, network system and connectivity, and that all network systems perform comparably with no major differences. One difference worth noticing between the network systems and using a direct connection is that sending data using a direct connection uses no kernel space CPU time, while sending data using the network systems uses a combination of kernel space and user space CPU time. This difference can possibly be explained by the former using direct memory access from the application sending data to the physical network card, while the latter has to go through the TUN/TAP kernel driver in order to process the packets.

None of the evaluated systems overload the CPU in testbeds A to D, but the CPU is overloaded in testbed E, i.e., the handheld test. The Nokia N900 is resource constrained, and even direct connections (when DevCom is not running) exhaust its CPU, shown on the left side of Fig. 11e. This overload is one possible explanation for its prevailing high latency and low throughput in the experiments described above. Sources of CPU utilization in DevCom include the cryptographic functions, but other functions, such as routing decisions, also affect CPU consumption.

An interesting observation is that GroupVPN, N2N and UIP consistently spend more CPU time in user space than in kernel space, and that P2PVPN, SocialVPN and DevCom do the opposite. The cryptographic functions occur in user space, while the TUN/TAP driver resides in kernel space. Furthermore, generating the workload is also affecting kernel space CPU utilization.

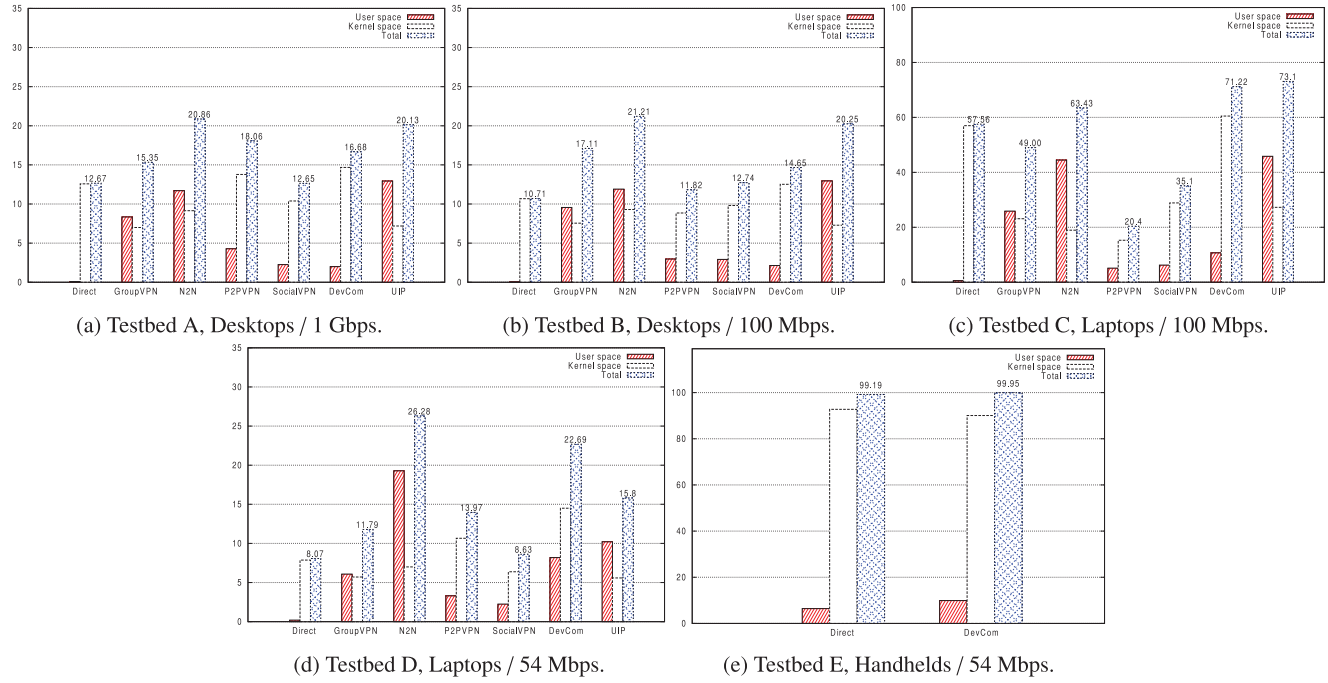
One anomaly visible in testbed C is that P2PVPN and SocialVPN have a lower average CPU consumption than direct connections. This could be attributed to high variance or outliers in the direct connection measurements, but when examining the raw samples this does not seem to be the case. It is possible than an external kernel thread has influenced the CPU consumption in the direct connection experiment. The average CPU consumption of direct connections in testbeds A, B and D, is between 8% and 13%.

It is interesting to see that the DevCom processing overhead is satisfactory considering that it has the highest throughput and lowest latency of all evaluated systems. In summary, the CPU overhead DevCom exhibits is expected and is in most cases insignificant, allowing other applications to run unaffected on the same device as DevCom.

5.2.4. Quality of experience

The micro-benchmarks show that the DevCom overhead is low, and an A/B QoE test intends to investigate if the overhead is noticeable to real users. The test is performed by installing a first-person shooter game, Quake III Arena, that is sensitive to latency on four identical desktop machines, i.e., two identical testbed A

[6] dd if = /dev/urandom | nc -[4 or 6] [IP] -q 1000 & sar -u ALL 1 1000.



(a) Testbed A, Desktops / 1 Gbps. (b) Testbed B, Desktops / 100 Mbps. (c) Testbed C, Laptops / 100 Mbps.

(d) Testbed D, Laptops / 54 Mbps. (e) Testbed E, Handhelds / 54 Mbps.

Fig. 11. CPU utilization results. The *y*-axis is in percent.

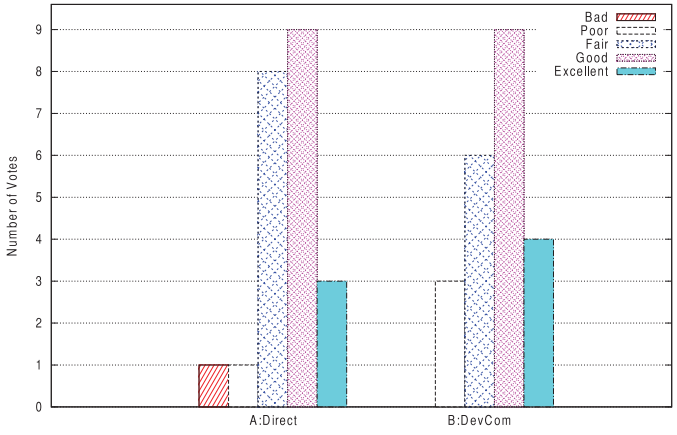


Fig. 12. A/B QoE test answers from 22 players.

configurations. DevCom is running on two of the machines (labeled B), while the other two communicate directly using the physical network (labeled A).

22 users played the game and answered questions regarding the game experience. The players ranked themselves as gamers (6 newbies, 12 casual gamers and 4 hardcore/pro gamers), stated their preference for machines A, B or neither, rated the game experience from 1 (Bad) to 5 (Excellent) on both A and B, and gave a brief statement about the difference in game experience on A and B, if any. Neither the players nor the person conducting the experiment knew about DevCom or the purpose of the experiment, making the test double-blind.

The answers of the players reveal that eight players prefer the A machines, ten players prefer the B machines and four players are unable to detect any difference in game experience. Fig. 12 shows the opinion scores and the corresponding number of votes for both A and B machines. It is visible that the B machines, where DevCom runs, have no players indicating a bad game experience and four players with an excellent game experience. The A machines, using direct connections, have one player with a bad experience and three players with an excellent game experience. As it is impossible to get better performance than with the physical network, the answers indicate that users are either imagining a difference, because the experience is equal, or that their experience is influenced by other factors. Mouse sensitivity, the game character's field-of-view and screen brightness are commented by players to be different, but all hardware and software configurations are verified to be identical on all four machines. The A/B test is not a full QoE evaluation, but substantiates the results from the micro-benchmarks and provides some insight about the user experience when using DevCom. DevCom allows applications to run on top of it with a high QoE for its users.

5.2.5. Connecting time

The connecting time is the period between when a device starts to connect to a device community and when that device has established control channels to all the other devices in the community. The connecting time is sensitive to parameters such as the number of devices in the device community, the processing power (partially because establishing the control channels involve several cryptography operations), and end-to-end delay between member devices. To get an indication of how DevCom behaves in real connecting scenarios an experiment is performed using a distributed device community consisting of five devices. The testbed setup is shown in Fig. 8 as testbed F. The physical network interface of the Dell Latitude laptop is restarted 20 times, and when network connectivity is available, the Dell Latitude establishes control channels to the four other devices, one by one. The order of control channel establishment is 1st) Desktop, 2nd) Media Center, 3rd) Laptop, and 4th) Laptop, also illustrated in Fig. 8. The connecting times are measured, and the cumulative results are shown in Table 7. Because of time constraints, this experiment does not cover situations where devices are behind NAT, which would negatively affect the connecting times.

It takes 48 ms in the best case scenario, 333 ms on average and 1045 ms in the worst case scenario to connect to the device community and establish connections to all the other members. Once

Table 7
Cumulative connecting times.

	Minimum	Average	Maximum	Std. Ddviation
1st	5 ms	5 ms	6 ms	0.2 ms
2nd	21 ms	160 ms	926 ms	330 ms
3rd	37 ms	175 ms	947 ms	330 ms
4th	48 ms	333 ms	1045 ms	441 ms

a connection is established it can be used immediately, so after 175 ms on average, the connecting device can communicate with three of the four other device community members.

The first connection is established significantly faster than the other three, taking only 6 ms in the worst case. One reason for this is that the first connection is established based on the list of historic physical addresses in the connecting device. In this experiment, the connecting device contacts the powerful desktop computer in close physical proximity first.

The connecting time experiment shows that DevCom has low connecting time which facilitates persistent connections without long interruptions in mobile scenarios. When a device using DevCom changes network, e.g., from Wi-Fi to 3G, all sessions and data transfers resume automatically and without user interaction within the connecting time.

The results are verified with an audio streaming test where one device streams audio to another device and no buffering is applied. The physical network interface of one device is taken down and brought up repeatedly, and the receiving device experiences only short interruptions corresponding to the connecting times of the sending device.

A possible optimization strategy that can reduce noticeable connecting times is prioritizing certain devices in the list of addresses where a connecting device attempts to connect. Connecting time is only relevant for applications with active communication channels, and connecting to the active devices first asserts that the noticeable connecting time is lowered.

5.3. Critical analysis of user-friendliness

The user-friendliness of DevCom is analyzed from the viewpoint of a novice, an intermediate user, and an application developer.

Only a single step is required if a novice user wants to join an existing device community, i.e., performing the initial join. It can be done with any of the techniques listed above, but using NFC or selecting the device community announcement in a user interface is probably most appropriate. After the initial join, DevCom takes care of the rest. This means that the devices can communicate, share and collaborate, despite barriers, such as NAT and changing addresses.

If no suitable device community exists, it is also necessary to create one, but this is trivial even for a novice, because creating a device community is achieved simply by choosing an arbitrary name. A default suggestion can be the user's own name in order to indicate that it is a personal device community, but any name is possible.

It is also possible to evict a device from a device community if it is misplaced, stolen or does not need to be part of a device community for some other reason. The user must choose the device and device community from a list and confirm the eviction, but DevCom automatically removes the public keys, closes all connections to the device and distributes the eviction information to all the other device community members.

A more interested user might want to have many devices working together in multiple device communities. This requires a bit more planning, i.e., choosing recognizable device community names and thinking more about which devices belong in which device communities. Furthermore, the initial join must be repeated for each device, but the process is the same and can be done incrementally.

Advanced users might want to use firewalls to allow an application in some device communities, but not in all. This can be done with existing firewall applications in the same way as with normal Internet traffic, meaning that the required steps only depend on the specific firewall application. Some firewalls have simple step-by-step wizards that guide the user when choosing applications and networks, while others have more features and require special knowledge.

Developing applications for DevCom is done in the same way as regular network applications. The same APIs are used and no special attention is needed in order to be DevCom compliant, beside supporting IPv6. That is why all types of existing applications already work with DevCom.

However, it is possible to integrate applications with DevCom if developers desire. The virtual network card has a unique and identifiable name, and the generated addresses have a rigid structure enabling easy extraction of, e.g., the device community names. For example, DevCom integration can enable built-in device community authorization where an application is only available to devices in a predefined device community, e.g., web-cam streaming is only allowed to devices in the `webcam` device community.

It is apparent that DevCom requires few steps to start functioning, and that its use is completely transparent. Users are not burdened by having to define IP ranges, choose dedicated super peers or manage cryptographic keys, and applications are not even aware that DevCom exists. Furthermore, application developers can also be oblivious to DevCom and continue to create network applications without any concern for DevCom. The benefits, such as persistent connections in advent of network changes and NAT and firewall penetration, are all automatically enabled by DevCom.

6. Conclusions

This paper presents DevCom, a new way to organize trustworthy groups of devices and the resources, data, and services they host. The two dominating approaches to support communication, sharing and collaboration today are LANs and VPNs, and compared to such solutions DevCom provides several advantages. One important drawback of both LANs and VPNs is that a device can only be member of a single group at a time. People are members of several communities, e.g., family, friends and work, and they most likely have different trust relations to the various communities. If a user is part of a corporate VPN today, all traffic goes through the company's network regardless of whether it is destined for another company device or for a friend's device. Multiple networks are essential in ubiquitous computing to allow seamless communication, sharing, and collaboration between disparate devices, and at the same time allow differentiated trust levels. To come one step closer to truly ubiquitous computing, social communities and interactions should be reflected and supported on the technical level. DevCom enables this with minimal user interaction, because nearly all tasks are performed automatically. The novel self-configuration mechanism introduced in DevCom can also be used in other zero configuration networking services where permanent, but unique addresses are needed. Furthermore, DevCom enables applications to run uninterrupted when devices change networks and obtain new addresses. This type of seamless connection handover is useful for nearly all types of mobile networking applications.

The combination of existing solutions for NAT penetration and cryptography algorithms for security and privacy protection has been shown to provide both trustworthiness and user-friendliness at the same time. It is also shown that all types of existing applications and services work with DevCom. The performance evaluation

of the prototype implementation demonstrates that the overhead is so low that it is unnoticeable for users, even for highly interactive applications such as first-person shooter games.

In addition to the advantages presented here, DevCom opens up new dimensions for researchers and developers of ubiquitous applications that can leverage the new features DevCom provides. This includes, for example, research in sensor networks for privacy protected home care and distributed application migration. It would also be interesting to see how the DevCom concepts can be applied in larger scale communities with many thousand devices, and if existing overlay systems, such as DHTs, can be used to aid such a transition.

Acknowledgments

The authors would like to thank Lea-Kathrin Kaese for conducting the A/B test and the participants for playing; Good game! We would also like to thank Stein Kristiansen for valuable feedback.

References

[1] M. Weiser, The computer for the 21st century, Sci. Am. 3 (3) (1991) 3–11.
[2] B.A. Ford, Unmanaged Internet Protocol: Taming the edge network management crisis, ACM SIGCOMM Comput. Commun. Rev. 34 (1) (2005) 93–98.
[3] LogMeIn, Inc., Hamachi, (World Wide Web). https://secure.logmein.com/products/hamachi/.
[4] B.C. Popescu, B. Crispo, A.S. Tanenbaum, Safe and private data sharing with turtle: friends team-up and beat the system, in: Proceedings of the 12th International Conference on Security Protocols, SP, 2006, pp. 213–220.
[5] T. Isdal, M. Piatek, A. Krishnamurthy, T. Anderson, Privacy-preserving P2P data sharing with OneSwarm, in: Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM, 2010, pp. 111–122.
[6] M.J. Freedman, R. Morris, Tarzan: a peer-to-peer anonymizing network layer, in: Proceedings of the 9th ACM conference on Computer and Communications Security, CCS, 2002, pp. 193–206.
[7] M. Waldman, D. Mazières, Tangler: a censorship-resistant publishing system based on document entanglements, in: Proceedings of the 8th ACM Conference on Computer and Communications Security, CCS, 2001, pp. 126–135.
[8] J. Frankel, WASTE P2P Darknet, 2003, (World Wide Web). http://waste.sourceforge.net/.
[9] OpenVPN Technologies, Inc, OpenVPN, (World Wide Web). http://openvpn.net/.
[10] C. Perkins, IP Mobility Support for IPv4, Revised, 2010, (World Wide Web). http://www.ietf.org/rfc/rfc5944.txt.
[11] C. Perkins, D. Johnson, J. Arkko, Mobility Support in IPv6, 2011, (World Wide Web). http://www.ietf.org/rfc/rfc6275.txt.
[12] P.S. Juste, D. Wolinsky, P.O. Boykin, M.J. Covington, R.J. Figueiredo, SocialVPN: enabling wide-area collaboration with integrated social and overlay networks, Comput. Networks 54 (12) (2010) 1926–1938.
[13] P.S. Juste, K. Jeong, H. Eom, C. Baker, R.J.O. Figueiredo, TinCan: user-defined P2P virtual network overlays for ad-hoc collaboration, EAI Endorsed Trans. Collab. Comput. (2014).
[14] L. Deri, R. Andrews, N2N: a layer two peer-to-peer VPN, in: Proceedings of the 2nd International Conference on Autonomous Infrastructure, Management and Security: Resilient Networks and Services, AIMS, 2008, pp. 53–64.
[15] W. Ginolas, Aufbau eines virtuellen privaten netzes mit peer-to-peer-technologie, Fachhochschule Wedel, 2009 (Master's thesis).
[16] S. Aoyagi, M. Takizawa, M. Saito, H. Aida, H. Tokuda, ELA: a fully distributed VPN system over peer-to-peer network, in: Proceedings of the 2005 Symposium on Applications and the Internet, SAINT, 2005, pp. 89–92.
[17] A. Ganguly, A. Agrawal, P.O. Boykin, R. Figueiredo, IP over P2P: enabling self-configuring virtual IP networks for grid computing, in: Proceedings of the 20th International Conference on Parallel and Distributed Processing, IPDPS, IEEE Computer Society, 2006.
[18] D. Wolinsky, K. Lee, P. Boykin, R. Figueiredo, On the design of autonomic, decentralized VPNs, in: 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010, pp. 1–10.
[19] B.A. Ford, UIA: A global connectivity architecture for mobile personal devices, Massachusetts Institute of Technology, 2008 (Ph.D. thesis).
[20] O. Hondaa, H. Ohsakia, M. Imasea, M. Ishizukab, J. Murayamab, Understanding TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency, in: M. Atiquzzaman, S.I. Balandin (Eds.), Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III, SPIE, 2005.
[21] S. Cheshire, M. Krochmal, Multicast DNS, 2013a, (World Wide Weba). http://www.ietf.org/rfc/rfc6762.txt.
[22] S. Cheshire, M. Krochmal, DNS-Based Service Discovery, 2013b, (World Wide Webb). http://www.ietf.org/rfc/rfc6763.txt.
[23] M. Yevmenkin, F. Thiel, Universal TUN/TAP device drive, 2000, (World Wide Web). https://www.kernel.org/doc/Documentation/networking/tuntap.txt.
[24] M. Krasnyansky, M. Yevmenkin, Universal TUN/TAP driver, 2001, (World Wide Web). http://vtun.sourceforge.net/tun/.
[25] The Avahi Team, Avahi, (World Wide Web). http://avahi.org/.
[26] The OpenSSL Project, OpenSSL: The Open Source toolkit for SSL/TLS, (World Wide Web). http://www.openssl.org/.
[27] DroidVPN, Tun.ko repository database, 2013, (World Wide Web). http://droidvpn.com/tun-repository.php.
[28] L. McVoy, The splice I/O model (1998). http://www.bitmover.com/lm/papers/splice.ps.